LSF JobScheduler Administrator’s Guide

Second Edition, August 1998

Platform Computing Corporation

LSF JobScheduler Administrator's Guide

Printed in Canada

Revision Information for LSF JobScheduler Administrator's Guide

Edition	Description
First	This document describes LSF JobScheduler 3.1, based on the *LSF Administrator's Guide,* Fourth Edition.
Second	Revised for LSF JobScheduler 3.2.

Contents

Preface

Audience

This guide describes the configuration and administration of LSF JobScheduler. It has been written for LSF administrators who manage LSF JobScheduler, although users who wish to understand the operational details of LSF JobScheduler will also find it useful.

This guide assumes you have knowledge of common system administration tasks such as exporting and mounting Network File System (NFS) partitions.

It is assumed that you have already read the LSF Installation Guide and installed LSF JobScheduler at your site.

LSF Suite 3.2

LSF is a suite of workload management products including the following:

LSF Batch is a batch job processing system for distributed and heterogeneous environments, which ensures optimal resource sharing.

LSF JobScheduler is a distributed production job scheduler that integrates heterogeneous servers into a virtual mainframe or virtual supercomputer

LSF MultiCluster supports resource sharing among multiple clusters of computers using LSF products, while maintaining resource ownership and cluster autonomy.

LSF Analyzer is a graphical tool for comprehensive workload data analysis. It processes cluster-wide job logs from LSF Batch and LSF JobScheduler to produce statistical reports on the usage of system resources by users on different hosts through various queues.

LSF Parallel is a software product that manages parallel job execution in a production networked environment.

LSF Make is a distributed and parallel Make based on GNU Make that simultaneously dispatches tasks to multiple hosts.

LSF Base is the software upon which all the other LSF products are based. It includes the network servers (LIM and RES), the LSF API, and load sharing tools.

There are two editions of the LSF Suite:

LSF Enterprise Edition

Platform's LSF Enterprise Edition provides a reliable, scalable means for organizations to schedule, analyze, and monitor their distributed workloads across heterogeneous UNIX and Windows NT computing environments. LSF Enterprise Edition includes all the features in LSF Standard Edition (LSF Base and LSF Batch), plus the benefits of LSF Analyzer and LSF MultiCluster.

LSF Standard Edition

The foundation for all LSF products, Platform's Standard Edition consists of two products, LSF Base and LSF Batch. LSF Standard Edition offers users robust load sharing and sophisticated batch scheduling across distributed UNIX and Windows NT computing environments.

Related Documents

The following guides are available from Platform Computing Corporation:

LSF Installation Guide
LSF Batch Administrator's Guide
LSF Batch Administrator's Quick Reference
LSF Batch User's Guide

LSF Batch User's Quick Reference
LSF JobScheduler Administrator's Guide
LSF JobScheduler User's Guide
LSF Analyzer User's Guide
LSF Parallel User's Guide
LSF Programmer's Guide

Online Documentation

- Man pages (accessed with the `man` command) for all commands
- Online help available through the Help menu for the `xlsbatch`, `xbmod`, `xbsub`, `xbalarms`, `xbcal` and `xlsadmin` applications.

Technical Assistance

If you need any technical assistance with LSF, please contact your reseller or Platform Computing's Technical Support Department at the following address:

LSF Technical Support
Platform Computing Corporation
3760 14th Avenue
Markham, Ontario
Canada L3R 3T7

Tel: +1 905 948 8448
Toll-free: 1-87PLATFORM (1-877-528-3676)
Fax: +1 905 948 9975
Electronic mail: *support@platform.com*

Please include the full name of your company.

You may find the answers you need from Platform Computing Corporation's home page on the World Wide Web. Point your browser to *www.platform.com*.

If you have any comments about this document, please send them to the attention of LSF Documentation at the address above, or send email to *doc@platform.com*.

1. Concepts

LSF is a suite of workload management products that schedule, monitor and analyze the workload of a network of computers. LSF JobScheduler allows you to schedule your mission-critical jobs across the whole network as if you were using a single mainframe computer.

LSF JobScheduler consists of a set of daemons that provide workload management services across the whole cluster, an API that allows access to such services at the procedure level, and a suite of tools or utilities that end users can use to access such services at the command or GUI level.

This chapter introduces important concepts related to the administration and operation of LSF JobScheduler. You should also read the *LSF JobScheduler User's Guide* to understand the concepts involved in using LSF JobScheduler.

Definitions

This section contains definitions of terms used in this guide.

Clusters

A *cluster* is a group of hosts that provides shared computing resources. Hosts can be grouped into clusters in a number of ways. A cluster can contain:

- all the hosts in a single administrative group
- all the hosts on one file server or sub-network
- hosts which perform similar functions

If you have hosts of more than one type, it is often convenient to group them together in the same cluster. LSF JobScheduler allows you to use these hosts transparently, so applications that run on only one host type are available to the entire cluster.

Submission, Master, and Execution Hosts

When LSF JobScheduler runs a job, three hosts are involved. The host from which the job is submitted is the *submission host*. The job information is sent to the *master host*, which is the host where the master LIM and `mbatchd` are running. The job is run on the *execution host*. It is possible for more than one of these to be the same host.

The master host is displayed by the `lsid` command:

```
% lsid
LSF 3.1, Dec 11, 1997
Copyright 1992-1997 Platform Computing Corporation

My cluster name is test_cluster
My master name is hostA
```

The following example shows the submission and execution hosts for a batch job:

```
hostD% bsub sleep 60
Job <1502> is submitted to default queue <normal>.

hostD% bjobs 1502
JOBID USER  STAT QUEUE  FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
1502  user2 RUN  normal hostD     hostB     sleep 60 Nov 22 14:03
```

The master host is *hostA*, as shown by the `lsid` command. The submission host is *hostD*, and the execution host is *hostB*.

Fault Tolerance

LSF JobScheduler has a number of features to support fault tolerance. LSF JobScheduler can tolerate the failure of any host or group of hosts in the cluster.

The LSF master host is chosen dynamically. If the current master host becomes unavailable, another host takes over automatically. The master host selection is based on the order in which hosts are listed in the `lsf.cluster.`*`cluster`* file. If the first host in the file is available, that host acts as the master. If the first host is unavailable, the second host takes over, and so on. LSF may be unavailable for a few minutes while hosts wait to be contacted by the new master. If the cluster is partitioned by a network failure, there will be a master on each side of the partition.

Fault tolerance in LSF JobScheduler depends on the event log file, `lsb.events`. Every event in the system is logged in this file, including all job submissions and job and host status changes. If the master host becomes unavailable, a new master is chosen by the LIMs. The slave daemon `sbatchd` on the new master starts a new master batch daemon `mbatchd`. The new `mbatchd` reads the `lsb.events` file to recover the state of the system.

If the network is partitioned, only one of the partitions can access the `lsb.events` log, so services are only available on one side of the partition. A lock file is used to guarantee that only one `mbatchd` is running in the cluster.

Running jobs are managed by the `sbatchd` on each server host. When the new `mbatchd` starts up it polls the `sbatchd` daemons on each host and finds the current status of its jobs. If `sbatchd` fails but the host is still running, jobs running on the host are not lost. When `sbatchd` is restarted it regains control of all jobs running on the host.

If an LSF server host fails, jobs running on that host are lost. No other jobs are affected.

If all of the hosts in a cluster go down, all running jobs are lost. When a host comes back up and takes over as master, it reads the `lsb.events` file to get the state of all jobs. Jobs that were running when the systems went down are assumed to have exited, and email is sent to the submitting user. Pending jobs remain in their queues, and are scheduled as hosts become available.

Shared Directories and Files

LSF is designed for networks where all hosts have shared file systems, and files have the same names on all hosts. LSF supports the Network File System (NFS), the Andrew

File System (AFS), and DCE's Distributed File System (DFS). NFS file systems can be mounted permanently or on demand using `automount`.

LSF includes support for copying user data to the execution host before running a job, and for copying results back after the job executes. In networks where the file systems are not shared, this can be used to give remote jobs access to local data.

For more information about running LSF on networks where no shared file space is available, see *'Using LSF JobScheduler without Shared File Systems'* on page 5.

Shared User Directories

To provide transparent remote execution, LSF commands determine the user's current working directory and use that directory on the remote host. For example, if the command `cc file.c` is executed remotely, `cc` only finds the correct `file.c` if the remote command runs in the same directory.

LSF JobScheduler automatically creates a `.lsbatch` subdirectory in the user's home directory on the execution host. This directory is used to store temporary input and output files for jobs.

Executables and the PATH Environment Variable

Search paths for executables (the PATH environment variable) are passed to the remote execution host unchanged. In mixed clusters, LSF works best when the user binary directories (`/usr/bin`, `/usr/local/bin`, etc.) have the same path names on different host types. This makes the PATH variable valid on all hosts.

If your user binaries are NFS-mounted, place all binaries in a shared file system under `/usr/local/lsf/mnt` (or some similar name), and then make a symbolic link from `/usr/local/bin` to `/usr/local/lsf/mnt/`*`type`*`/bin` for the correct host type on each machine. This is what LSF's Default installation procedure does.

LSF configuration files are normally in a shared directory. This makes administration easier. There is little performance penalty for this, because the configuration files are not read often.

For more information on LSF installation directories, see the *LSF Installation Guide.*

Using LSF JobScheduler without Shared File Systems

Some networks do not share files between hosts. LSF JobScheduler can still be used on these networks, with reduced fault tolerance.

You must choose one host to act as the LSF JobScheduler master host. The LSF JobScheduler configuration files and working directories must be installed on this host, and the master host must be listed first in the `lsf.cluster.`*`cluster`* file.

If the master host is unavailable, users cannot submit batch jobs or check job status. Running jobs continue to run, but no new jobs are started. When the master host becomes available again, LSF JobScheduler service is resumed.

Some fault tolerance can be introduced by choosing more than one host as possible master hosts, and using NFS to mount the LSF JobScheduler working directory on only these hosts. All the possible master hosts must be listed first in the `lsf.cluster.`*`cluster`* file. As long as one of these hosts is available, LSF JobScheduler continues to operate.

Resources and Resource Requirements

LSF provides a powerful means for you to describe your heterogeneous cluster in terms of resources. One of the most important decisions LSF makes when scheduling a job is to map a job's resource requirements onto resources available on individual hosts. There are several types of resource. *Load indices* measure dynamic resource availability such as a host's CPU load or available swap space. *Static resources* represent unchanging information such as the number of CPUs a host has, the host type, and the maximum available swap space.

Resources may also be described in terms of where they are located. A *shared resource* is a resource which is associated with the entire cluster or a subset of hosts within the cluster. In contrast to host-based resources such as memory or swap space, using a shared resource from one machine affects the availability of that resource as seen by other machines. Common examples of shared resources include floating licenses for software packages, shared file systems, and network bandwidth. LSF provides a mechanism to configure which machines share a particular resource and to monitor the availability of those resources.

Resource names may be any string of characters, excluding the characters reserved as operators. The `lsinfo` command lists the resources available in your cluster.

For a complete description of resources and how they are used, see *Chapter 4, 'Resources', on page 45* of the *LSF JobScheduler User's Guide*.

To best place a job for optimized performance, resource requirements can be specified for each application. A resource requirement is an expression that contains resource names and operators. Resource requirements can be configured for individual applications, or specified for each job. The detailed format for resource requirements can be found in *'Resource Requirement Strings'* on page 51of the *LSF JobScheduler User's Guide.*

Remote Execution Control

There are two aspects to controlling access to remote execution. The first requirement is to authenticate the user. When a user executes a remote command, the command must be run with that user's permission. The LSF daemons need to know which user is requesting the remote execution. The second requirement is to check access controls on the remote host. The user must be authorized to execute commands remotely on the host.

User Authentication Methods

LSF supports user authentication using privileged ports, authentication using the RFC 931 or RFC 1413 identification protocols, and site-specific external authentication, such as Kerberos and DCE.

The default method is to use privileged ports. To use privileged ports, some of the LSF utilities must be installed with `root` as the owner of the file and with the `setuid` bit set.

UNIX

Authentication Using Privileged Ports

If a load-sharing program is owned by `root` and has the `setuid` bit set, the LSF API functions use a privileged port to communicate with LSF servers, and the servers accept the user ID supplied by the caller. This is the same user authentication mechanism as used by `rlogin` and `rsh`.

When a `setuid` application calls the `LSLIB` initialization routine, a number of privileged ports are allocated for remote connections to LSF servers. The effective user ID then reverts to the real user ID. Therefore, the number of remote connections is limited. Note that an LSF utility reuses the connection to the RES for all remote task executions on that host, so the number of privileged ports is only a limitation on the number of remote hosts that can be used by a single application, not on the number of remote tasks. Programs using `LSLIB` can specify the number of privileged ports to be created at initialization time.

Authentication Using Identification Daemons

The RFC 1413 and RFC 931 protocols use an identification daemon running on each client host. Using an identification daemon incurs more overhead, but removes the need for LSF applications to allocate privileged ports. All LSF commands except `lsadmin` can be run without `setuid` permission if an identification daemon is used.

You should use identification daemons if your site cannot install programs owned by root with the `setuid` bit set, or if you have software developers creating new load-sharing applications in C using `LSLIB`.

An implementation of RFC 931 or RFC 1413, such as `pidentd` or `authd`, may be obtained from the public domain (if you have access to Internet FTP, a good source for identification daemons is host `ftp.lysator.liu.se`, directory `pub/ident/servers`). RFC 1413 is a more recent standard than RFC 931. LSF is compatible with either.

External Authentication

You can configure your own user authentication scheme using the `eauth` mechanism of LSF. If external authentication is used, an executable called `eauth` must be written and installed in `LSF_SERVERDIR`.

When an LSF client program is invoked (e.g., `lsrun`), the client program automatically executes `eauth -c` *`hostname`* to get the external authentication data. *hostname* is the name of the host running the LSF daemon (e.g., RES). The external user authentication data can be passed to LSF via `eauth`'s standard output.

When the LSF daemon receives the request, it executes `eauth -s` under the primary LSF administrator user ID. The parameter, LSF_EAUTH_USER, must be configured in the `/etc/lsf.sudoers` file if your site needs to run authentication under another user ID (see *'The lsf.sudoers File'* on page 89 for details). `eauth -s` is executed to process the user authentication data. The data is passed to `eauth -s` via its standard input. The standard input stream has the following format:

```
uid gid username client_addr client_port user_auth_data_len
user_auth_data
```

The variables are listed below:

- `uid` is the user ID in ASCII of the client user
- `gid` is the group ID in ASCII of the client user
- `username` is the user name of the client user
- `client_addr` is the host address of the client host in ASCII dot notation
- `client_port` is the port number from where the client request is made
- `user_auth_data_len` is the length of the external authentication data in ASCII passed from the client
- `user_auth_data` is the external user authentication data passed from the client

The LSF daemon expects `eauth -s` to write `1` to its standard output if authentication succeeds, or `0` if authentication fails.

The same `eauth -s` process can service multiple authentication requests; if the process terminates, the LSF daemon will re-invoke `eauth -s` on the next authentication request.

Example uses of external authentication include support for Kerberos 4 and DCE client authentication using the GSSAPI. These examples can be found in the `examples/krb` and `examples/dce` directories in the standard LSF distribution. Installation instructions are found in the README file in these directories.

Security of LSF Authentication

All authentication methods supported by LSF depend on the security of the `root` account on all hosts in the cluster. If a user can get access to the `root` account, they can subvert any of the authentication methods. There are no known security holes that allow a non-`root` user to execute programs with another user's permission.

Some people have particular concerns about security schemes involving RFC 1413 identification daemons. When a request is coming from an unknown host, there is no way to know whether the identification daemon on that host is correctly identifying the originating user.

LSF only accepts job execution requests that originate from hosts within the LSF cluster, so the identification daemon can be trusted. The identification protocol uses a port in the UNIX privileged port range, so it is not possible for an ordinary user to start a hacked identification daemon on an LSF host.

NT

LSF Security

The default authentication of method of LSF is to use privileged ports. On UNIX, this requires binaries which need to be authenticated (for example `bsub`) to be made `setuid root`. NT does not have the concept of `setuid` binaries and does not restrict which binaries can use privileged ports. A security risk exists that a user can discover the format of LSF protocol messages and write a program which tries to communicate with an LSF server. It is recommended that external authentication (via `eauth`) be used where this security risk is a concern.

The system environment variable LSF_ENVDIR is used by LSF to obtain the location of `lsf.conf` which points to important configuration files. Any user who can modify system environment variables can modify LSF_ENVDIR to point to their own configuration and start up programs under the `lsfadmin` account.

Once the LSF service is started, it will only accept requests from the `lsfadmin` account. To allow other users to interact with the LSF service, you must set up the `lsf.sudoers` file under the directory specified by the SYSTEMROOT environment variable. See *'The lsf.sudoers File'* on page 89 for the format of the `lsf.sudoers` file.

Note

Only the LSF_STARTUP_USERS and LSF_STARTUP_PATH are used on NT. You should ensure that only authorized users modify the files under the SYSTEMROOT directory.

All external binaries invoked by the LSF daemons (such as `esub`, `eexec`, `elim`, `eauth`, and queue level pre- and post-execution commands) are run under the `lsfadmin` account.

How LSF Chooses Authentication Methods

LSF uses the `LSF_AUTH` parameter in the `lsf.conf` file to determine the type of authentication to use.

If an LSF application is not `setuid` to `root`, library functions use a non-privileged port. If the `LSF_AUTH` flag is not set in the `lsf.conf` file, the connection is rejected. If `LSF_AUTH` is defined to be `ident`, the RES on the remote host, or `mbatchd` in the case of a `bsub` command, contacts the identification daemon on the local host to verify the user ID. The identification daemon looks directly into the kernel to make sure the network port number being used is attached to a program being run by the specified user.

LSF allows both the `setuid` and authentication daemon methods to be in effect simultaneously. If the effective user ID of a load-sharing application is `root`, then a privileged port number is used in contacting the RES. RES always accepts requests from a privileged port on a known host even if `LSF_AUTH` is defined to be `ident`. If the effective user ID of the application is not `root`, and the `LSF_AUTH` parameter is defined to be `ident`, then a normal port number is used and RES tries to contact the identification daemon to verify the user's identity.

External user authentication is used if `LSF_AUTH` is defined to be `eauth`. In this case, LSF will run the external executable `eauth` in the `LSF_SERVERDIR` directory to do the authentication.

The error message "`User permission denied`" is displayed by `lsrun`, `bsub`, and other LSF commands if LSF cannot verify the user's identity. This may be because the LSF applications are not installed `setuid`, the NFS directory is mounted with the `nosuid` option, the identification daemon is not available on the local or submitting host, or the external authentication failed.

If you change the authentication type while the LSF daemons are running, you will need to run the command `lsfdaemons start` on each of the LSF server hosts so that the daemons will use the new authentication method.

Host Authentication Methods

When a batch job or a remote execution request is received, LSF first determines the user's identity. Once the user's identity is known, LSF decides whether it can trust the host from which the request comes from.

Trust LSF Host

LSF normally allows remote execution by all users except `root`, from all hosts in the LSF cluster, i.e. LSF trusts all hosts that are configured into your cluster. The reason for this is that by configuring an LSF cluster, you are turning a network of machines into a single computer. Users must have valid accounts on all hosts. This allows any user to run a job with their own permission on any host in the cluster. Remote execution requests and batch job submissions are rejected if they come from a host not in the LSF cluster.

A site can configure an external executable to perform additional user or host authorization. By defining `LSF_AUTH` to be `eauth`, the LSF daemon will invoke `eauth -s` when it receives a request that needs authentication and authorization. As an example, this `eauth` can check if the client user is on a list of authorized users.

UNIX

Using `/etc/hosts.equiv`

If the `LSF_USE_HOSTEQUIV` parameter is set in the `lsf.conf` file, LSF uses the same remote execution access control mechanism as the `rsh` command. When a job is run on a remote host, the user name and originating host are checked using the `ruserok(3)` function on the remote host.

This function checks in the `/etc/hosts.equiv` file and the user's `$HOME/.rhosts` file to decide if the user has permission to execute jobs.

The name of the local host should be included in this list. RES calls `ruserok()` for connections from the local host. `mbatchd` calls `ruserok()` on the master host, so every LSF JobScheduler user must have a valid account and remote execution permission on the master host.

The disadvantage of using the `/etc/hosts.equiv` and `$HOME/.rhosts` files is that these files also grant permission to use the `rlogin` and `rsh` commands without giving a password. Such access is restricted by security policies at some sites.

See the `hosts.equiv(5)` and `ruserok(3)` manual pages for details on the format of the files and the access checks performed.

The error message "`User permission denied`" is displayed by `lsrun`, `bsub`, and other LSF commands if you configure LSF to use `ruserok()` and the client host is not found in either the `/etc/hosts.equiv` or the `$HOME/.rhosts` file on the master or remote host.

User Account Mapping

By default, LSF assumes uniform user accounts throughout the cluster. This means that job will be executed on any host with exactly the same user ID and user login name.

LSF JobScheduler has a mechanism to allow user account mapping across dissimilar name spaces. Account mapping can be done at the individual user level. Individual users of the LSF cluster can set up their own account mapping by setting up a `.lsfhosts` file in their home directories.

The LSF administrator can disable user account mapping.

How LSF JobScheduler Schedules Jobs

LSF JobScheduler provides the functions of a traditional mainframe job scheduler with transparent operation across a network. Jobs can be submitted and monitored from anywhere in the LSF cluster.

LSF's master scheduler, `mbatchd`, exists in the network, accepting client requests such as job submissions, job modifications, and controls. It also dispatches jobs to run on all LSF server hosts when the jobs are ready to run. There is one slave scheduler, `sbatchd`, that lives on every host configured as an LSF JobScheduler server. Each `sbatchd` accepts jobs dispatched from the `mbatchd`, runs them on the local host, and monitors the jobs that are running.

There is a Load Information Manager, LIM, running on every host, that collects and propagates resource and load information. This information is then provided to `mbatchd` to help choose the most appropriate host for running a job.

When a job is submitted to LSF JobScheduler, it enters a queue. The job remains pending in the queue until all its required conditions are met. Many factors control when and where the job should run:

- resource requirements of the job
- resource availability in the whole LSF cluster
- time conditions associated with the job (such as a calendar)
- inter-job dependencies
- file status events
- external events
- load conditions on all hosts

`mbatchd` periodically scans through the jobs that are ready to run. In doing so, it first obtains load and resource information from the LIM. When appropriate resources become available for the job, LSF JobScheduler compares the load conditions of all qualified hosts and runs the job on a host that does the job best. The job is sent to the `sbatchd` on the most appropriate host via a TCP/IP connection.

When `sbatchd` receives a job from `mbatchd`, it initializes the job's execution environment first. Such initialization can be customized to fit the user's preference. By default, all user environment variables from the submission host are automatically copied to the execution host and re-established. The user can also choose to reinitialize the environment on the execution host at job submission time.

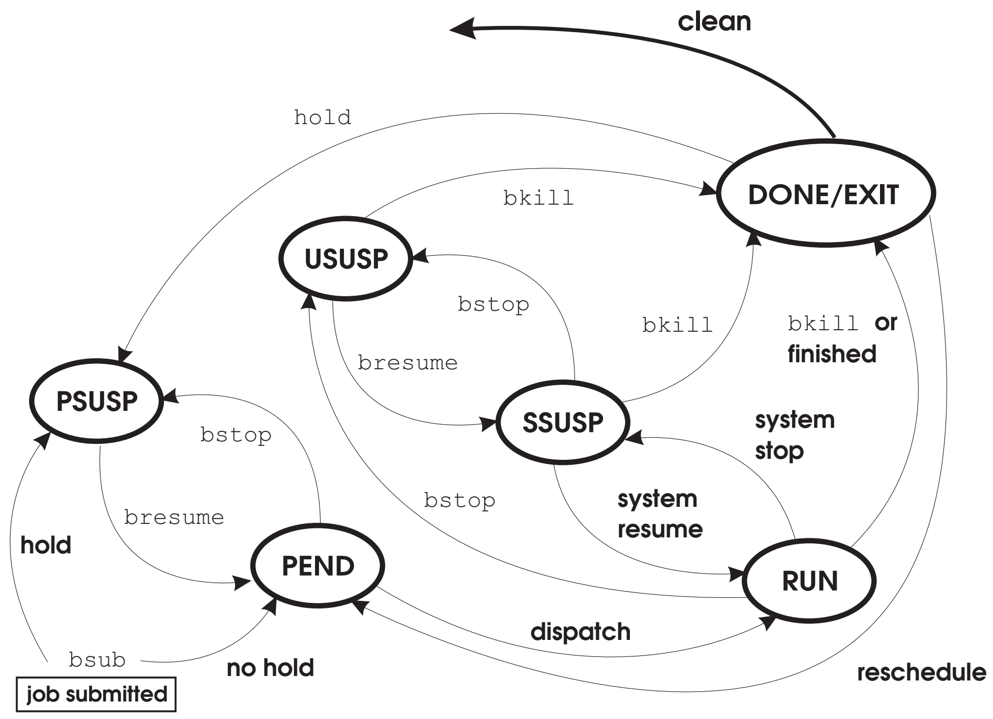
A job starter can be configured by the cluster administrator to start the job in a certain environment, such as a particular shell.

When the job is started, `sbatchd` keeps track of all processes of the job through the Process Information Manager, PIM. The resource consumption information of the whole job is periodically sampled and passed to `mbatchd`. When the job finishes, `sbatchd` reports the status back to `mbatchd`.

Job States

An LSF JobScheduler job goes through a series of state transitions until it eventually completes its task, fails or is terminated. The possible states of a job during its life cycle are shown in *Figure 1*.

Figure 1. Job States



Many jobs enter only three states:

PEND - waiting in the queue

RUN - dispatched to a host and running

DONE - terminated normally

Pending

A job remains in the PEND state until all conditions for its execution are met. Some of the conditions are:

- dependency on a calendar
- dependency on another job
- dependency on external events such as file status
- availability of the specified resources

The `bjobs -lp` command displays the reason why a job is currently in the PEND state.

Terminated

A job may terminate abnormally for various reasons. Job termination may happen from any state. An abnormally terminated job goes into `EXIT` state. The situations where a job terminates abnormally include:

- the job is cancelled by the user while pending, or after being started
- the job fails to start successfully, e.g. the wrong executable is specified by the user when the job is submitted
- the job exits with a non-zero exit status

A job that has terminated may return to the PEND state once again if it is a repetitive job, and if it is not waiting for the next time schedule.

Suspended

Jobs may also be suspended at any time. A job can be suspended by its owner, by the LSF administrator, by the `root` user (superuser), or by LSF JobScheduler. There are three different states for suspended jobs:

PSUSP

The job was suspended by its owner or the LSF administrator while in `PEND` state (a job can also be in the PSUSP state if it was submitted with the `hold` option).

USUSP

The job was suspended by its owner or the LSF administrator after being dispatched.

SSUSP

The job was suspended by LSF JobScheduler after being dispatched.

The `bjobs -s` command displays the reason why a job was suspended.

User Suspended

A job may be suspended by its owner or the LSF administrator with the `bstop` command. These jobs are considered user-suspended (displayed by `bjobs` as USUSP).

When the user restarts the job with the `bresume` command, the job is not resumed immediately to prevent overloading. Instead, the job is changed from USUSP to SSUSP (suspended by the system). The SSUSP job is resumed when the host load levels are within the scheduling thresholds for that job, exactly as for jobs suspended because of high load.

Pre- and Post-Execution Commands

Each job can be associated with optional pre- and post-execution commands.

If a pre-execution command is specified, the job is held in the queue until the specified pre-execution command returns a successful exit status (zero). While the job is pending, other jobs may go ahead of the waiting job.

If a post-execution command is specified, then the command is run after the job is finished.

Pre- and post-execution commands are arbitrary command lines.

Pre-execution commands can be used to support job starting decisions which cannot be configured directly in LSF JobScheduler.

Post-execution commands are typically used to clean up some state left by the pre-execution and the job execution.

LSF JobScheduler supports both job level and queue level pre-execution. Post-execution is only supported at the queue level.

See *'Queue-Level Pre-/Post-Execution Commands'* on page 99 for more information about queue-level pre / post-execution commands, and *'Pre-execution Commands'* on page 72 of the *LSF JobScheduler User's Guide* for more information about the job-level pre-execution commands.

LSF Daemons

LSF consists of:

- a number of daemons running on each of the server machines providing workload management services across the cluster
- an API that allows clients to access the services
- a set of user interfaces that allow end users to submit, monitor and control the workload to the cluster

In order to effectively manage LSF JobScheduler, it is important to understand the operation of these daemons.

Load Information Manager (LIM)

LIM is a key server that forms the basis for the concept of an LSF cluster. LIM belongs to LSF Base and provides a single system image called a cluster.

LIM runs on every server host and provides cluster configuration information, load information, and host selection services. The LIMs on all hosts coordinate in collecting and transmitting load and resource information. Load information is transmitted between LIMs in the form of vectors of load indices.

All LIMs in a cluster must share the same cluster configuration in order to work correctly. To run LIM on a host, the host must be configured as a server host in the `lsf.cluster.`*`cluster`* file. See *'The lsf.cluster.cluster File'* on page 86 for more information.

A master LIM is elected for each LSF cluster. The master LIM receives load information from all slave LIMs and provides services to all hosts. The slave LIMs periodically check their own load conditions and send a load vector to the master if significant changes in load condition are observed. The minimum load information exchange interval is 15 seconds. Slave LIMs also monitor the status of the master LIM and elect a new master if the original one becomes unavailable. This provides high availability because as long as one host is up, the master will be up for services.

The load indices monitored at a site can be extended by directing the LIM to invoke and communicate with an External Load Information Manager (ELIM). The ELIM is responsible for collecting load indices not managed by the LIM. These indices are passed on to the LIM by ELIM through a well-defined protocol.

Remote Execution Server (RES)

RES runs on every machine that runs jobs through LSF. RES provides remote execution and remote file operation services. LSF JobScheduler uses RES to do file transfers across machines. RES is also used to run interactive jobs. Together with LIM, RES belongs to LSF Base.

Master Batch Daemon (`mbatchd`)

`mbatchd` runs on the host where master LIM is running. There can be only one `mbatchd` per LSF cluster. `mbatchd` is the job scheduler daemon that schedules jobs according to user defined schedules as well as system configured policies. `mbatchd` gets resource and load information from the master LIM and chooses the most appropriate host to run a job that is ready. Powerful mechanisms are built into `mbatchd` to provide intelligent and configurable job scheduling functions and reliable operations in cases of failures.

`mbatchd` works closely with the slave batch daemon (`sbatchd`) in coordinating job executions.

`mbatchd` is always automatically started by `sbatchd` on the master host.

Slave Batch Daemon (`sbatchd`)

`sbatchd` is the execution daemon for all batch jobs. Jobs that are ready to run are dispatched to `sbatchd`s from the `mbatchd`. Together with each job are job specifications which `sbatchd` uses to run the job and control its execution. `sbatchd` initiates the job according to its specifications and monitors the job throughout the job's life time. When the job finishes, `sbatchd` reports the job status back to `mbatchd`. The `sbatchd` on the master host is responsible for the starting of `mbatchd` on that host.

Alarm Daemon (`alarmd`)

`alarmd` is the alarm daemon for LSF JobScheduler. `alarmd` is a daemon started by `mbatchd` and is used to perform periodic operations on the alarm log file, `lsb.alarms.log`, including sending renotifications and moving resolved or expired alarm incidents into the alarm history file, `lsb.alarms.hist`. The alarm log and history files are stored in LSB_SHAREDIR/`logdir`.

Alarm incidents are appended to the alarm log through the `raisealarm` command which is invoked by `mbatchd` when an alert condition happens. `alarmd` reads the alarm definition in the `lsb.alarms` file to determine the method for sending renotifcations. See *'The lsb.alarms File'* on page 101 for details of the alarm configuration file. The `alarmd` is started whenever `mbatchd` is started and exits when it detects that `mbatchd`, which started it, is not running. It runs under the user account of the primary LSF administrator.

External Event Daemon (`eeventd`)

LSF has an open system architecture to allow each site to customize the behaviour of the system. External events are site specific conditions that can be used to trigger job scheduling actions. Examples of external events are data arrival, tape silo status, and exceptional conditions. External events are collected by the External Event Daemon (`eeventd`). The `eeventd` runs on the same host as the `mbatchd` and collects site specific events that LSF JobScheduler will use to trigger the scheduling of jobs. LSF

JobScheduler comes with a default `eeventd` that monitors file events. A user site can easily add more event functions to it to monitor more events.

For more details see *‘External Event Management’* on page 68.

Remote File Access

When LSF JobScheduler runs a job, it attempts to run the job in the directory where the `bsub` command was invoked. If the execution directory is under the user’s home directory, `sbatchd` looks for the path relative to the user’s home directory.

If the directory is not available on the execution host, the job is run in `/tmp`. Any files created by the job, including the standard output and error files, are left on the execution host.

LSF provides support for moving user data from the submission host to the execution host before executing a job, and from the execution host back to the submitting host after the job completes. The file operations can be specified when submitting a job.

The LSF JobScheduler remote file access mechanism uses `lsrcp(1)` to process the file transfer. `lsrcp` first tries to connect to the RES daemon on the submission host to handle the file transfer. If `lsrcp` cannot contact the RES on the submission host, it attempts to use `rcp` to copy the file. You must set up the `/etc/hosts.equiv` or `HOME/.rhosts` file in order to use `rcp`. See the `rcp(1)` and `rsh(1)` manual pages for more information on using `rcp`.

A site may replace `lsrcp` with its own file transfer mechanism as long as it supports the same syntax as `lsrcp(1)`. This may be done to take advantage of a faster interconnection network or to overcome limitations with the existing `lsrcp`. `sbatchd` looks for the `lsrcp` executable in the `LSF_BINDIR` directory as specified in the `lsf.conf` file.

For a complete description of the LSF remote file access facilities, see the `bsub(1)` manual page and the *LSF JobScheduler User’s Guide*.

2. Managing LSF Base

This chapter describes the operation, maintenance and tuning of an LSF Base cluster. The correct operation of LSF Base is essential to LSF JobScheduler. This chapter should be read by all LSF cluster administrators.

Managing Error Logs

Error logs contain important information about daemon operations. When you see any abnormal behavior related to any of the LSF daemons, you should check the relevant error logs to find out the cause of the problem.

LSF log files grow over time. These files should occasionally be cleared, either by hand or using automatic scripts. You can also define a calendar-driven job to do the cleanup regularly.

LSF JobScheduler Daemon Error Log

All LSF log files are reopened each time a message is logged, so if you rename or remove a log file of an LSF daemon, the daemons will automatically create a new log file.

The LSF daemons log messages when they detect problems or unusual situations. The daemons can be configured to put these messages into files.

UNIX The daemons can also be configured to send error messages to the system error logs using the `syslog` facility.

If `LSF_LOGDIR` is defined in the `lsf.conf` file, LSF daemons try to store their messages in files in that directory. Note that `LSF_LOGDIR` must be writable by root.

The error log file names for the LSF Base system daemons, LIM and RES, are `lim.log.`*`hostname`*, and `res.log.`*`hostname`*.

The error log file names for LSF JobScheduler daemons are `sbatchd.log.`*`hostname`*, `mbatchd.log.`*`hostname`*, `pim.log.`*`hostname`*, and `eeventd.log.`*`hostname`*.

LSF daemons log error messages in different levels so that you can choose to log all messages or only log messages that are critical enough. This is controlled by parameter `LSF_LOG_MASK` in the `lsf.conf`. Possible values for this parameter are discussed in '*LSF_LOG_MASK*' on page 80.

UNIX

If `LSF_LOGDIR` is defined but the daemons cannot write to files there, the error log files are created in `/tmp`.

If `LSF_LOGDIR` is not defined, then errors are logged to `syslog` using the `LOG_DAEMON` facility. `syslog` messages are highly configurable, and the default configuration varies widely from system to system. Start by looking for the file `/etc/syslog.conf`, and read the manual pages for `syslog` and/or `syslogd`.

If LSF daemons cannot find the `lsf.conf` file when they start, they will not find the definition of `LSF_LOGDIR`. In this case, error messages go to syslog. If you cannot find any error messages in the log files, they are likely in the syslog.

NT

If `LSF_LOGDIR` is defined but the daemons cannot write to files there, the error log files are created in `C:\TEMP`.

LSF_LOGDIR must be defined, or all error messages will be lost.

FLEXlm Log

The FLEXlm license server daemons log messages about the state of the license servers, and when licenses are checked in or out. This log helps to resolve problems with the license servers and to track license use.

The FLEXlm log is configured by the `lsflicsetup` command. This log file grows over time. You can remove or rename the existing FLEXlm log file at any time. The script `lsf_license` uses to run the FLEXlm daemons creates a new log file when necessary.

Note

If you already have FLEXlm server running for other products and LSF JobScheduler licenses are added to the existing license file, then the log messages for FLEXlm should go to the same place as you previously set up for other products.

Controlling LIM and RES Daemons

The LSF cluster administrator can monitor the status of the hosts in a cluster, start and stop the LSF daemons, and reconfigure the cluster. Many operations are done using the `lsadmin` command, which performs administrative operations on LSF Base daemons, LIM and RES.

Checking Host Status

The `lshosts` and `lsload` commands report the current status and load levels of hosts in an LSF cluster. The `lsmon` and `xlsmon` commands provide a running display of the same information. The LSF administrator can find unavailable or overloaded hosts with these tools.

```
% lsload
HOST_NAME  status  r15s   r1m  r15m   ut    pg   ls   it   tmp   swp   mem
hostD          ok   1.3   1.2   0.9  92%   0.0    2   20    5M  148M   88M
hostB         -ok   0.1   0.3   0.7  0%    0.0    1   67   45M   25M   34M
hostA        busy   8.0  *7.0   4.9  84%   4.6    6   17    1M   81M   27M
```

When the status of a host is proceeded by a '-', it means RES is not running on that host. In the above example, RES on *hostB* is down.

Restarting LIM and RES

LIM and RES can be restarted to upgrade software or clear persistent errors. Jobs running on the host are not affected by restarting the daemons. The LIM and RES daemons are restarted using the `lsadmin` command:

```
% lsadmin

lsadmin>limrestart hostD

Checking configuration files ...
No errors found.

Restart LIM on <hostD> ...... done
lsadmin>resrestart hostD
Restart RES on <hostD> ...... done
lsadmin>quit
```

Note

You must login as the LSF cluster administrator to run the `lsadmin` *command.*

The `lsadmin` command can be applied to all available hosts by using the host name `all`, as follows:

```
% lsadmin limrestart all
```

If a daemon is not responding to network connections `lsadmin` displays an error message with the host name. In this case you must kill and restart the daemon by hand.

Remote Startup of LIM and RES

LSF administrators can start up any, or all, LSF daemons, on any, or all, LSF hosts, from any host in the LSF JobScheduler cluster. For this to work, file `/etc/lsf.sudoers` has to be set up properly to allow you to start up daemons as root and you should be able to run `rsh` across LSF hosts without having to enter a password. See *'The lsf.sudoers File'* on page 89 for configuration details of `lsf.sudoers`.

The '`limstartup`' and '`resstartup`' options in `lsadmin` allow for the startup of the LIM and RES daemons respectively. Specifying a host name allows for starting up a daemon on particular host. For example,

```
% lsadmin limstartup hostA
Starting up LIM on <hostA> ...... done

% lsadmin resstartup hostA
Starting up RES on <hostA> ...... done
```

The `lsadmin` command can be used to start up all available hosts by using the host name '`all`'; for example, '`lsadmin limstartup all`'. All LSF daemons, including LIM, RES, and `sbatchd`, can be started on all LSF hosts using the command `lsfstartup`.

Shutting down LIM and RES

All LSF daemons can be shut down at any time. If the LIM daemon on the current master host is shut down, another host automatically takes over as master. If the RES daemon is shut down while remote interactive tasks are running on the host, the running tasks continue but no new tasks are accepted. To shutdown LIM and RES, use `lsadmin` command:

```
% lsadmin

lsadmin>limshutdown hostD
Shut down LIM on <hostD> ...... done

lsadmin>resshutdown hostD
Shut down RES on <hostD> ...... done

lsadmin>quit
```

You can run `lsadmin reconfig` while the LSF system is in use; users may be unable to submit new jobs for a short time, but all current remote executions are unaffected.

Managing LSF Configuration

Overview of LSF Configuration Files

LSF configuration consists of several levels:

`lsf.conf`

This is the generic LSF environment configuration file. This file defines general installation parameters so that all LSF executables can find the necessary information. This file is typically installed in the directory in which all LSF server binaries are installed, and a symbolic link is made from a convenient directory as defined by the environment variable `LSF_ENVDIR`, or the default directory `/etc`. This file is created by the `lsfsetup` during installation. Note that many of the parameters in this file are machine specific. Detailed contents of this file are described in *'The lsf.conf File'* on page 75.

LIM Configuration Files

LIM is the kernel of your cluster that provides the single system image to all applications. LIM reads the LIM configuration files and determines your cluster and the cluster master host.

LIM files include `lsf.shared`, and `lsf.cluster.`*`cluster`*, where *cluster* is the name of your LSF JobScheduler cluster. These files define the host members, general host attributes, and resource definitions for your cluster. The individual functions of each of the files are described below.

`lsf.shared` defines the available resource names, host types, host models, cluster names and external load indices that can be used by all clusters. This file is shared by all clusters.

`lsf.cluster.`*`cluster`* file is a per cluster configuration file. It contains two types of configuration information: cluster definition information and LIM policy information. Cluster definition information impacts all LSF applications, while LIM policy information impacts applications that rely on LIM's policy for job placement.

The cluster definition information defines cluster administrators, all the hosts that make up the cluster, attributes of each individual host such as host type, host model, and resources using the names defined in `lsf.shared`.

LIM policy information defines the load sharing and job placement policies provided by LIM. More details about LIM policies are described in *'Controlling LIM and RES Daemons'* on page 25.

LIM configuration files are stored in directory `LSF_CONFDIR` as defined in `lsf.conf` file. Details of LIM configuration files are described in *'The lsf.shared File'* on page 83.

LSF JobScheduler Configuration Files

These files define LSF JobScheduler specific configuration such as queues and server hosts. These files are only read by `mbatchd`. The LSF JobScheduler configuration relies on LIM configuration. LSF JobScheduler daemons get the cluster configuration information from the LIM via the LSF API.

LSF JobScheduler configuration files are stored in directory `LSB_CONFDIR/`*`cluster`*, where LSB_CONFDIR is defined in `lsf.conf`, and *cluster* is the name of your cluster. Details of LSF JobScheduler configuration files are described in *Section 5, 'LSF JobScheduler Configuration Reference', beginning on page 93*.

Configuration File Formats

All configuration files except `lsf.conf` use a section-based format. Each file contains a number of sections. Each section starts with a line beginning with the reserved word `Begin` followed by a section name, and ends with a line beginning with the reserved word `End` followed by the same section name. `Begin`, `End`, section names and keywords are all case insensitive.

Sections can either be vertical or horizontal. A horizontal section contains a number of lines, each having the format: `keyword = value`, where value is one or more strings. For example:

```
Begin exampleSection
key1 = string1
key2 = string2 string3
```

```
key3 = string4
End exampleSection

Begin exampleSection
key1 = STRING1
key2 = STRING2 STRING3
End exampleSection
```

In many cases you can define more than one object of the same type by giving more than one horizontal section with the same section name.

A vertical section has a line of keywords as the first line. The lines following the first line are values assigned to the corresponding keywords. Values that contain more than one string must be bracketed with '(' and ')'. The above examples can also be expressed in one vertical section:

```
Begin exampleSection
key1     key2               key3
string1  (string2 string3)  string4
STRING1  (STRING2 STRING3)  -
End exampleSection
```

Each line in a vertical section is equivalent to a horizontal section with the same section name.

Some keys in certain sections are optional. For a horizontal section, an optional key does not appear in the section if its value is not defined. For a vertical section, an optional keyword must appear in the keyword line if any line in the section defines a value for that keyword. To specify the default value use '-' or '()' in the corresponding column, as shown for `key3` in the example above.

Each line may have multiple columns, separated by either spaces or TAB characters. Lines can be extended by a '\' (back slash) at the end of a line. A '#' (pound sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted. Blank lines are ignored.

Example Configuration Files

Below are some examples of LIM configuration and LSF JobScheduler configuration files. Detailed explanations of the variables are given in *Section 4, 'LSF Base Configuration Reference', beginning on page 75.*

Sample `lsf.shared` file

```
Begin Cluster
ClusterName                          # This line is keyword(s)
test_cluster
End Cluster

Begin HostType
TYPENAME                             # This line is keyword(s)
hppa
SUNSOL
sgi
rs6000
alpha
NTX86
End HostType

Begin HostModel
MODELNAME          CPUFACTOR          # This line is keyword(s)
HP735              4.0
DEC3000            5.0
ORIGIN2K           8.0
PENTI120           3.0
End HostModel

Begin Resource
RESOURCENAME        DESCRIPTION        #This line is keyword(s)
hpux                (HP-UX operating system)
decunix             (Digital Unix)
solaris             (Sun Solaris operating system)
NT                  (Windows NT operating system)
fserver             (File Server)
cserver             (Compute Server)
End Resource
```

Sample `lsf.cluster.test_cluster` file:

```
Begin ClusterManager
Manager = lsf user7
End ClusterManager

Begin Host
HostName   Model      Type     server    swp    Resources
hostA      HP735      hppa     1         2     (fserver hpux)
hostD      ORIGIN2K   sgi      1         2     (cserver)
hostB      PENT200    NTX86    1         2     (NT)
End Host
```

Changing LIM Configuration

This section gives procedures for some common changes to the LIM configuration. There are different ways for you to change LIM configuration:

- using `lsfsetup` as described in *‘Configuration File Formats’* on page 29
- editing individual files using an editor

The following procedures focus on changing configuration files using an editor so that you can understand the concepts behind the configuration changes.

Adding a Host to a Cluster

To add a host to an existing LSF JobScheduler cluster, use the following procedure.

Step 1 If you are adding a host of a new host type, make sure you do the steps described in *‘Installing an Additional Host Type’* on page 73 of the *LSF Installation Guide* first.

Step 2 If you are adding a host of a type for which you already installed LSF binaries, make sure that the LSF binaries, configuration files, and working directories are NFS-mounted on the new host. For each new host you add, follow the host setup procedure as described in *‘Adding an Additional Host to an Existing Cluster’* on page 79 in the *LSF Installation Guide*.

Step 3 If you are adding a new host type to the cluster, modify the “HostType” section of the `lsf.shared` file to add the new host type. A host type can be any alphanumeric string up to 29 characters long.

Step 4 If you are adding a new host model, modify the “HostModel” section of your `lsf.shared` file to add in the new model together with its CPU speed factor relative to other models.

Step 5 For each host you add into the cluster, you should add a line to the “Host” section of the `lsf.cluster.`*`cluster`* file with host name, host type, and all other attributes defined, as shown in *‘Example Configuration Files’* on page 31.

The master LIM and `mbatchd` daemons run on the first available host in the “Host” section of your `lsf.cluster.`*`cluster`* file, so you should list reliable batch server hosts first. For more information, see *‘Fault Tolerance’* on page 5.

If you are adding a client host, set the SERVER field for the host to `0` (zero).

Step 6 Reconfigure your LSF JobScheduler cluster so that LIM knows that you have added a new host to the cluster. Follow the instructions in *‘Reconfiguring an LSF Cluster’* on page 39. If you are adding more than one host, do this step after you have done step 1 to 5 for all added hosts.

Step 7 If you are adding hosts as LSF JobScheduler server hosts, add these hosts to LSF JobScheduler configuration by following steps described in *‘Restarting sbatchd’* on page 56.

Step 8 Start the LSF daemons on the newly added host(s) by running the following command, and using `ps` to make sure that `res`, `lim` and `sbatchd` have started:

```
LSF_SERVERDIR/lsf_daemons start
```

CAUTION!

LSF daemons start must be run as `root`. If you are creating a private cluster, do not attempt to use `lsf_daemons` to start your daemons. Start them manually.

Removing Hosts From a Cluster

To remove a host from an existing LSF JobScheduler cluster, use the following procedure.

Step 1 If you are running LSF JobScheduler, make sure you remove unwanted hosts from the LSF JobScheduler first following steps described in *'Restarting sbatchd'* on page 56.

Step 2 Edit your `lsf.cluster.`*`cluster`* file and remove the unwanted hosts from the "Host" section.

Step 3 Log in to any host in the cluster as the LSF JobScheduler administrator. Run the following command:

```
lsadmin resshutdown  host1 host2 ...
```

Here, `host1`, `host2`, ... are hosts you want to remove from your cluster.

Step 4 Follow instructions in *'Reconfiguring an LSF Cluster'* on page 39 to reconfigure your LSF JobScheduler cluster. The LIMs on the removed hosts will quit upon reconfiguration.

UNIX **Step 5** Remove the LSF section from the host's system startup files. This undoes what you have done previously to start LSF daemons at boot time. See *'Starting LSF Servers at Boot Time'* on page 86 in the *LSF Installation Guide* for details.

Host Resources

Your cluster is most likely heterogeneous. Even if your computers are all the same, it may still be heterogeneous. For example, some machines are configured as file servers, while others are compute servers; some have more memory, others have less, some have four CPUs, others have only one; some have host-locked software licenses installed, others do not. LSF JobScheduler provides powerful resource selection mechanisms so that correct hosts with required resources are chosen to run your jobs.

Customizing Host Resources

For maximum flexibility, you should characterize your resources clearly enough so that users have enough choices. For example, if some of your machines are connected to both Ethernet and FDDI, while others are only connected to Ethernet, then you probably want to define a resource called `fddi` and associate the `fddi` resource to machines connected to FDDI. This way, users can specify resource `fddi` if they want their jobs to run on machines connected to FDDI.

To customize host resources for your cluster, use the following procedure.

Step 1 Log in to any host in the cluster as the LSF JobScheduler administrator.

Step 2 Define new resource names by modifying the “Resource” section of the `lsf.shared` file. Add a brief description to each of the added resource names. Resource descriptions will be displayed to a user by `lsinfo` command.

Step 3 If you want to associate added resource names to an application, edit `lsf.task` file properly to reflect the resource into the resource requirements of the application. Alternatively, you can leave this to individual users who can use `lsrtasks` command to customize his / her own file.

Step 4 Edit the `lsf.cluster.`*`cluster`* file to modify the RESOURCES column of the “Host” section so that all hosts that have the added resources will now have the added resource names in that column.

Step 5 Follow instructions in ‘*Reconfiguring an LSF Cluster*’ on page 39 to reconfigure your LSF JobScheduler cluster.

Configuring Resources in LSF Base

Resources are defined in the “Resource” section of the `lsf.shared` file. The definition of a resource involves specifying a name and description, as well as, optionally, the type of its value, its update interval, and whether a higher or lower value indicates greater availability.

The mandatory resource information fields are:

- a **`RESOURCENAME`** indicating the name of the resource

- a **DESCRIPTION** that should indicate what the resource represents

The optional resource information fields are:

- a **TYPE** indicating its value (boolean, numeric or string)
- an **INTERVAL** indicating how often the value is updated (for resources whose value changes dynamically)
- an **INCREASING** flag indicating whether a higher value represents a greater availability of the resource (for numeric resources which can be used for scheduling jobs)

When the optional attributes are not specified, the resource is treated as static and boolean-valued.

The following is a sample of a "Resource" section from an `lsf.shared` file:

```
Begin Resource
RESOURCENAME  TYPE    INTERVAL  INCREASING  DESCRIPTION
mips          Boolean ()        ()          (MIPS architecture)
dec           Boolean ()        ()          (DECStation system)
sparc         Boolean ()        ()          (SUN SPARC)
hppa          Boolean ()        ()          (HPPA architecture)
bsd           Boolean ()        ()          (BSD unix)
sysv          Boolean ()        ()          (System V UNIX)
hpux          Boolean ()        ()          (HP-UX UNIX)
aix           Boolean ()        ()          (AIX UNIX)
nt            Boolean ()        ()          (Windows NT)
scratch       Numeric 30        N           (Shared scratch space on server)
synopsys      Numeric 30        N           (Floating licenses for Synopsys)
verilog       Numeric 30        N           (Floating licenses for Verilog)
console       String  30        N           (User Logged in on console)
End Resource
```

There is no distinction between shared and non-shared resources in the resource definition in the `lsf.shared` file.

Note

The NewIndex section in the `lsf.shared` *file is obsolete. To achieve the same effect, the "Resource" section of the* `lsf.shared` *file can be used to define a dynamic numeric resource, and the "default" keyword can be used in the LOCATION field of the "ResourceMap" section of the* `lsf.cluster.`*`cluster`* *file.*

Associating Resources with Hosts

Resources are associated with the host(s) on which they are available in the "ResourceMap" section of the `lsf.cluster.`*`cluster`* file (where *`cluster`* is the name of the cluster). The following fields must be completed for each resource:

- a RESOURCENAME indicating the name of the resource, as defined in the `lsf.shared` file
- a LOCATION indicating whether the resource is shared or non-shared, across which hosts, and with which initial value(s)

The following is an example of a "ResourceMap" section from an `lsf.cluster.`*`cluster`* file:

```
Begin ResourceMap
RESOURCENAME   LOCATION
verilog        5@[all]
synopsys       (2@[apple] 2@[others])
console        (1@[apple] 1@[orange])
End ResourceMap
```

The possible states of a resource that may be specified in the LOCATION column are:

- each host in the cluster has the resource
- the resource is shared by all hosts in the cluster
- there are multiple instances of a resource within the cluster, and each instance is shared by a unique subset of hosts

For static resources, the LOCATION column should contain the value of the resource.

The syntax of the information in the fields of the LOCATION column takes one of two forms. For static resources, where the value must be specified, use:

```
(value1@[host1 host2 ...] value2@[host3 host4] ...)
```

For dynamic resources, where the value is updated by an ELIM, use:

```
([host1 host2 ...] [host3 host4 ...] ...)
```

Each set of hosts listed within the square brackets specifies an instance of the resource. All hosts within the instance share the resource whose quantity is indicated by its value. In the above example, `host1, host2,...` form one instance of the resource, `host3, host4,...` form another instance and so on.

Note

The same host cannot be in more than one instance of a resource.

Three pre-defined words have special meaning in this specification:

- `all` refers to all the server hosts in the cluster, i.e., `value@[all]` means the resource is shared by all server hosts in the cluster made up of `host1 host2 ... hostn`
- `others` refers to the rest of the server hosts listed in the cluster, i.e. `(2@[apple] 2[others])` means there are 2 units of "syno" on apple, and 2 shared by all other hosts
- `default` refers to each host, i.e. `value@[default]` is equivalent to `(value@[host1] value@[host2] ... value@[hostn])` where `host1, ... hostn` are all server hosts in the cluster.

These syntax examples assume that static resources (requiring values) are being specified. For dynamic resources, use the same syntax but omit the `value`.

The following items should be taken into consideration when configuring resources under LSF Base.

In the `lsf.cluster.`*`cluster`* file, the “Host” section must precede the “ResourceMap” section since the “ResourceMap” section uses the hostnames defined in the “Host” section.

- The RESOURCES column in the “Host” section of the `lsf.cluster.`*`cluster`* file should be used to associate static boolean resources with particular hosts. Using the “ResourceMap” section for static boolean resources section will result in an empty RESOURCES column in the `lshosts(1)` display.

- All resources specified in the “ResourceMap” section are treated as shared resources which are displayed using the `lsload -s` or `lshosts -s` commands. The exception is for dynamic numeric resources specified using the `default` pre-defined word. These will be treated together with load indices such as 'mem' and 'swap' and are viewed using the `lsload -l` command.

If the “ResourceMap” section is not defined, then any dynamic resources specified in `lsf.shared` are considered to be host-based (i.e the resource is available on each host in the cluster).

Reconfiguring an LSF Cluster

After changing LIM configuration files you must tell LIM to read the new configuration. Use the `lsadmin` command to tell LIM to pick up the new configuration.

Operations can be specified on the command line or entered at a prompt. Run the `lsadmin` command with no arguments, and enter `help` to see the available operations.

The `lsadmin reconfig` command checks the LIM configuration files for errors. If no errors are found, the command confirms that you want to restart the LIMs on all hosts, and reconfigures all the LIM daemons:

```
% lsadmin reconfig
Checking configuration files ...
No errors found.

Do you really want to restart LIMs on all hosts? [y/n] y
Restart LIM on <hostD> ...... done
Restart LIM on <hostA> ...... done
Restart LIM on <hostC> ...... done
```

In the above example no errors are found. If any non-fatal errors are found, the command asks you to confirm the reconfiguration. If fatal errors are found, the reconfiguration is aborted.

If you want to see details on any errors, run the command `lsadmin ckconfig -v`. This reports all errors to your terminal.

If you change the configuration file of LIM, you should also reconfigure LSF JobScheduler by running `badmin reconfig` because LSF JobScheduler depends on LIM configuration. If you change the configuration of LSF JobScheduler, then you only need to run `badmin reconfig`.

External Resource Collection

The values of static external resources are specified through the `lsf.cluster.`*`cluster`* file. All dynamic resources, regardless of whether they are shared or host-based, are collected through an ELIM. An ELIM is started in the following situations:

- On every host if any dynamic resource is configured as host-based. For example, if the LOCATION field in the “ResourceMap” section of `lsf.cluster.`*`cluster`* is `([default])`, then every host will start an ELIM.
- On the master host for any cluster-wide resources. For example, if the LOCATION field in the “ResourceMap” section of `lsf.cluster.`*`cluster`* is `([all])`, then an ELIM is started on the master host.
- On the first host specified for each instance, if multiple instances of the resource exist within the cluster. For example, if the LOCATION field in the “ResourceMap” section of `lsf.cluster.`*`cluster`* is `([hostA hostB hostC] [hostD hostE hostF])`, then an ELIM will be started on hostA and hostD to report the value of that resource for that set of hosts.

  If the host reporting the value for an instance goes down, then an ELIM is started on the next available host in the instance. In the above example, if *hostA* became unavailable, an ELIM is started on *hostB*. If *hostA* becomes available again, then the ELIM on *hostB* is shut down and the one on *hostA* is started.

Note that a maximum of one ELIM is started on each host, regardless of the type of resources on which it reports. If only cluster-wide resources are being used, then an ELIM will only be started on the master host. In order to simply write a single ELIM for all hosts which reports on a combination of shared and non-shared resources, the following variables must be set in the ELIM's environment:

- LSF_MASTER: this variable is defined if the ELIM is being invoked on the master host. It is undefined otherwise. This can be used to test whether the ELIM should report on cluster-wide resources which only need to be collected on the master host.
- LSF_RESOURCES: this variable contains a list of resource names (separated by spaces) on which the ELIM is expected to report. A resource name is only put in the list if the host on which the ELIM is running belongs to an instance of that resource.

Restrictions

The following restrictions apply to the use of shared resources in LSF products.

- A shared resource cannot be used as a load threshold in the "Hosts" section of the `lsf.cluster.`*`cluster`* file.
- A shared resource cannot be used in the loadSched / loadStop thresholds, or in the STOP_COND or RESUME_COND parameters in the queue definition in the `lsb.queues` file.

Writing an External LIM

The ELIM can be any executable program, either an interpreted script or compiled code. Example code for an ELIM is included in the `misc` directory in the LSF distribution. The `elim.c` file is an ELIM written in C. You can customize this example to collect the load indices you want.

The ELIM communicates with the LIM by periodically writing a load update string to its standard output. The load update string contains the number of indices followed by a list of name-value pairs in the following format:

```
N name1 value1 name2 value2 ... nameN valueN
```

For example:

```
3 tmp2 47.5 nio 344.0 licenses 5
```

This string reports 3 indices: `tmp2`, `nio`, and `licenses`, with values 47.5, 344.0, and 5 respectively. Index values must be numbers between `-INFINIT_LOAD` and `INFINIT_LOAD` as defined in the `lsf.h` header file.

If the ELIM is implemented as a C program, as part of initialization it should use `setbuf(3)` to establish unbuffered output to `stdout`.

The ELIM should ensure that the entire load update string is written successfully to `stdout`. This can be done by checking the return value of `printf(3s)` if the ELIM is implemented as a C program or the return code of `/bin/echo(1)` from a shell script. The ELIM should exit if it fails to write the load information.

Each LIM sends updated load information to the master every 15 seconds. Depending on how quickly your external load indices change, the ELIM should write the load update string once every 15 seconds at most. If the external load indices rarely change, the ELIM can write the new values only when a change is detected. The LIM continues to use the old values until new values are received.

The executable for the ELIM must be in LSF_SERVERDIR and must have the name '`elim`'. If any external load indices are defined in the LIM configuration file, the LIM invokes the ELIM automatically on startup. The ELIM runs with the same user id and file access permission as the LIM.

The LIM restarts the ELIM if it exits; to prevent problems in case of a fatal error in the ELIM, it is restarted once every 90 seconds at most. When the LIM terminates, it sends a `SIGTERM` signal to the ELIM. The ELIM must exit upon receiving this signal.

Overriding Built-In Load Indices

The ELIM can also return values for the built-in load indices. In this case the value produced by the ELIM overrides the value produced by the LIM. The ELIM must ensure that the semantics of any index it supplies is the same as that of the corresponding index returned by the `lsinfo(1)` command.

For example, some sites prefer to use `/usr/tmp` for temporary files. To override the `tmp` load index, write a program that periodically measures the space in the `/usr/tmp` file system, and writes the value to standard output. Name this program `elim` and put it in the LSF_SERVERDIR directory.

Note

The name of an external load index must not be one of the resource name aliases `cpu`*,* `idle`*,* `logins`*, or* `swap`*. To override one of these indices, use its formal name:* `r1m`*,* `it`*,* `ls`*, or* `swp`*.*

You must configure the external load index even if you are overriding a built-in load index.

Tuning CPU Factors

CPU factors are used to differentiate the relative speed of different machines. LSF JobScheduler runs jobs on the best possible machines so that the response time is minimized. To achieve this, it is important that you define correct CPU factors for each machine model in your cluster by changing the “HostModel” section of your `lsf.shared` file.

CPU factors should be set based on a benchmark that reflects your work load. (If there is no such benchmark, CPU factors can be set based on raw CPU power.) The CPU factor of the slowest hosts should be set to one, and faster hosts should be proportional to the slowest. For example, consider a cluster with two hosts, `hostA` and `hostB`, where `hostA` takes 30 seconds to run your favourite benchmark and `hostB` takes 15 seconds to run the same test. `hostA` should have a CPU factor of 1, and `hostB` (since it is twice as fast) should have a CPU factor of 2.

LSF JobScheduler uses a normalized CPU performance rating to decide which host has the most available CPU power. The normalized ratings can be seen by running the `lsload -N` command. The hosts in your cluster are displayed in order from best to worst. Normalized CPU run queue length values are based on an estimate of the time it would take each host to run one additional unit of work, given that an unloaded host with CPU factor 1 runs one unit of work in one unit of time.

Incorrect CPU factors can reduce performance in two ways. If the CPU factor for a host is too low, that host may not be selected for job placement when a slower host is available. This means that jobs would not always run on the fastest available host. If the CPU factor is too high, jobs are run on the fast host even when they would finish sooner on a slower but lightly loaded host. This causes the faster host to be overused while the slower hosts are underused.

Both of these conditions are self-correcting to some extent. If the CPU factor for a host is too high, jobs are sent to that host until the CPU load threshold is reached. The LIM then marks that host as busy, and no further jobs will be sent there. If the CPU factor is too low, jobs may be sent to slower hosts. This increases the load on the slower hosts, making LSF JobScheduler more likely to schedule future jobs on the faster host.

LSF License Management

LSF software is licensed using the FLEXlm license manager from Globetrotter Software, Inc. The LSF license key controls the hosts allowed to run LSF.

UNIX The procedures for obtaining, installing and upgrading license keys are described in *'Getting License Key Information'* on page 34 of, and *'Setting Up the License Key'* on page 36 of the *LSF Installation Guide*.

NT Information on installing and upgrading license keys can be found in *'License Installation Options'* on page 56 of the *LSF Installation Guide*.

FLEXlm controls the total number of hosts configured in all your LSF clusters. You can organize your hosts into clusters in whatever fashion you choose. Each server host requires at least one license. Multiprocessor hosts require more than one, as a function of the number of processors. Each client host requires 1/5 of a license.

LSF uses two kinds of FLEXlm license: file-based DEMO licenses and server-based permanent licenses.

DEMO Licenses

The DEMO license allows you to try LSF out on an unlimited number of hosts on any supported host type. The trial period has a fixed expiry date, and the LSF software will not function after that date. DEMO licenses do not require any additional daemons.

Permanent Licenses

Permanent licenses are the most common. A permanent license limits only the total number of hosts that can run the LSF software, and normally has no time limit. You can choose which hosts in your network will run LSF, and how they are arranged into clusters. Permanent licenses are counted by a license daemon running on one host on your network.

For permanent licenses, you need to choose a license server host and send hardware host identification numbers for the license server host to your software vendor. The vendor uses this information to create a permanent license that is keyed to the license server host. Some host types have a built-in hardware host ID; on others, the hardware address of the primary LAN interface is used.

How FLEXlm Works

UNIX

FLEXlm is used by many UNIX software packages because it provides a simple and flexible method for controlling access to licensed software. A single FLEXlm license server can handle licenses for many software packages, even if those packages come from different vendors. This reduces the systems administration load, since you do not need to install a new license manager every time you get a new package.

The License Server Daemon

FLEXlm uses a daemon called `lmgrd` to manage permanent licenses. This daemon runs on one host on your network, and handles license requests from all applications. Each license key is associated with a particular software vendor. `lmgrd` automatically starts a *vendor daemon*; the LSF version is called `lsf_ld` and is provided by Platform Computing Corporation. The vendor daemon keeps track of all licenses supported by that vendor. DEMO licenses do not require you to run license daemons.

The license server daemons should be run on a reliable host, since licensed software will not run if it cannot contact the license server. The FLEXlm daemons create very little load, so they are usually run on the file server. If you are concerned about availability, you can run `lmgrd` on a set of three hosts. As long as a majority of the license server hosts are available, applications can obtain licenses.

The License File

The license file is named `license.dat`.

Location

Software licenses are stored in a text file.

UNIX The default location for this license file is `/usr/local/flexlm/licenses/license.dat`.

NT The default location for this license file is `c:\flexlm\license.dat`.

This may be different at your site, depending on what decisions were made when FLEXlm was initially installed.

The license file must be accessible from every host that runs licensed software. Normally, it is most convenient to place the license file in a shared directory. The variable LSF_LICENSE_FILE in the `lsf.conf` file should point to this location, allowing LSF to locate the license file.

Port@Host Configuration

An alternative to specifying a file pathname in the LSF_LICENSE_FILE variable is to use “port@host” notation to indicate the name of the license server host and port being used by the `lmgrd` daemon. For example:

```
LSF_LICENSE_FILE=”1700@hostD, 1700@hostC, 1700@hostB”
```

The port number must be the same as that specified in the license file.

Contents

The `license.dat` file normally contains:

- A SERVER line for each FLEXlm server host. The SERVER line contains the host name, hardware host ID and network port number for the server.

- A DAEMON line for each software vendor, which gives the file path name of the vendor daemon.
- A FEATURE line for each software license. This line contains the number of copies that may be run, along with other necessary information.

The FEATURE line contains an encrypted code to prevent tampering. For permanent licenses, the licenses granted by the FEATURE line can be accessed only through license servers listed on the SERVER lines.

For DEMO licenses no FLEXlm daemons are needed, so the license file contains only the FEATURE line.

Sample License Files

This sample DEMO license file contains one line for each separate product (see *'Modifying LSF Products and Licensing'* on page 49). However, no SERVER or DAEMON information is needed. The license is for LSF version 3.1 and is valid until Jun. 10, 1998.

```
FEATURE lsf_base lsf_ld 3.100 10-Jun-1998 0 5C51F231E238555BAD7F "Platform" DEMO
FEATURE lsf_jobscheduler lsf_ld 3.100 10-Jun-1998 0 6CC1D2C137651068E23C "Platform" DEMO
FEATURE lsf_jobscheduler_server lsf_ld 3.100 10-Jun-1998 0 6CC1D2C137651068E23C "Platform" DEMO
FEATURE lsf_multicluster lsf_ld 3.100 10-Jun-1998 0 2CC1F2E132C85B8D1806 "Platform" DEMO
```

In this sample permanent license file, the license server is configured to run on `hostD`, using TCP port 1700. This allows 10 hosts to run LSF, with no expiry date.

```
SERVER hostD 08000962cc47 1700 DAEMON lsf_ld /usr/local/lsf/etc/lsf_ld
FEATURE lsf_base lsf_ld 3.100 01-Jan-0000 0 51F2315CE238555BAD7F "Platform"
FEATURE lsf_jobscheduler lsf_ld 3.100 01-Jan-0000 0 C1D2C1376C651068E23C "Platform"
FEATURE lsf_jobscheduler_server lsf_ld 3.100 01-Jan-0000 0 C1D2C1376C651068E23C "Platform"
FEATURE lsf_multicluster lsf_ld 3.100 01-Jan-0000 0 C1F2E1322CC85B8D1806 "Platform"
```

UNIX License Management Utilities

FLEXlm provides several utility programs for managing software licenses. These utilities and their manual pages are included in the LSF software distribution.

Because these utilities can be used to shut down the FLEXlm license server, and thus prevent licensed software from running, they are installed in the

LSF_SERVERDIR directory. The file permissions are set so that only `root` and members of group 0 can use them.

The utilities included are:

`lmcksum` - calculate check sums of the license key information

`lmdown` - shut down the FLEXlm server

`lmhostid` - display the hardware host ID

`lmremove` - remove a feature from the list of checked out features

`lmreread` - tell the license daemons to re-read the license file

`lmstat` - display the status of the license servers and checked out licenses

`lmver` - display the FLEXlm version information for a program or library

For complete details on these commands, see the on-line manual pages.

Updating an LSF License

FLEXlm only accepts one license key for each feature listed in a license key file. If there is more than one FEATURE line for the same feature, only the first FEATURE line is used. To add hosts to your LSF cluster, you must replace the old FEATURE line with a new one listing the new total number of licenses.

UNIX The procedure for updating a license key file to include new license keys is described in '*Adding a Permanent License*' on page 40 of the *LSF Installation Guide*.

Changing the FLEXlm Server TCP Port

The fourth field on the SERVER line specifies the TCP port number that the FLEXlm server uses. Choose an unused port number. LSF usually uses port numbers in the range 3879 to 3882, so the numbers from 3883 on are good choices. If the `lmgrd` daemon complains that the license server port is in use, you can choose another port number and restart `lmgrd`.

For example, if your license file contains the line:

```
SERVER hostname host-id 1700
```

and you want your FLEXlm server to use TCP port 3883, change the SERVER line to:

```
SERVER hostname host-id 3883
```

Modifying LSF Products and Licensing

LSF Suite V3.1 includes the following products: LSF Base, LSF Batch, LSF JobScheduler, LSF MultiCluster, LSF Make, and LSF Analyzer.

The configuration changes to enable a particular product in a cluster are handled during installation by `lsfsetup`. If at some later time you want to modify the products in your cluster, edit the PRODUCTS line in the 'Parameters' section of the `lsf.cluster.`*`cluster`* file. You can specify any combination of the strings 'LSF_Base', 'LSF_Batch', 'LSF_JobScheduler', 'LSF_MultiCluster', and 'LSF_Analyzer'. If any of 'LSF_Batch', 'LSF_JobScheduler', or 'LSF_MultiCluster' are specified, then 'LSF_Base' is automatically enabled as well.

If the `lsf.cluster.`*`cluster`* file is shared, adding a product name to the PRODUCTS line enables that product for all hosts in the cluster. For example, enable the operation of LSF Base, LSF Batch and LSF MultiCluster:

```
Begin Parameters
PRODUCTS=LSF_Base LSF_Batch LSF_MultiCluster
End Parameters
```

Enable the operation of LSF Base only:

```
Begin Parameters
PRODUCTS=LSF_Base
End Parameters
```

Enable the operation of LSF JobScheduler:

```
Begin Parameters
PRODUCTS=LSF_JobScheduler
End Parameters
```

Selected Hosts

It is possible to indicate that only certain hosts run LSF Batch or LSF JobScheduler within a cluster. This is done by specifying 'LSF_Batch' or 'LSF_JobScheduler' in the RESOURCES field of the "Hosts" section of the `lsf.cluster.`*`cluster`* file. For example, the following enables hosts `hostA`, `hostB`, and `hostC` to run LSF JobScheduler and hosts `hostD`, `hostE`, and `hostF` to run LSF Batch.

```
Begin Parameters
PRODUCTS=LSF_Batch
End Parameters

Begin   Host
HOSTNAME   model   type      server RESOURCES
hostA      SUN41   SPARCSLC  1      (sparc bsd LSF_JobScheduler)
hostB      HPPA9   HP735     1      (linux LSF_JobScheduler)
hostC      SGI     SGIINDIG  1      (irix cs LSF_JobScheduler)
hostD      SUNSOL  SunSparc  1      (solaris)
hostE      HP_UX   A900      1      (hpux cs bigmem)
hostF      ALPHA   DEC5000   1      (alpha)
End Hosts
```

The license file used to serve the cluster must have the corresponding features. A host will show as unlicensed if the license for the product it was configured to run is unavailable. For example, if a cluster is configured to run LSF JobScheduler on all hosts, and the license file does not contain the LSF JobScheduler product, than the hosts will be unlicensed, even if there are licenses for LSF Base or LSF Batch.

3. Managing LSF JobScheduler

This chapter describes the operating concepts and maintenance tasks of LSF JobScheduler. This chapter requires concepts from '*Managing LSF Base*' on page 23. The topics covered in this chapter are:

- managing LSF JobScheduler logs
- controlling LSF JobScheduler servers
- controlling LSF JobScheduler queues
- managing LSF JobScheduler configuration
- controlling LSF JobScheduler jobs
- controlling job execution environment
- system calendars
- example LSF JobScheduler configuration files

Managing LSF JobScheduler Logs

Managing error log files for LSF JobScheduler daemons was described in '*Managing Error Logs*' on page 23. This section discusses the other important log files LSF JobScheduler daemons produce. The LSF JobScheduler log files are found in the directory `LSB_SHAREDIR/`*cluster*`/logdir`.

LSF JobScheduler Accounting Log

Each time a job completes or exits, an entry is appended to the `lsb.acct` file. This file can be used to create accounting summaries of LSF JobScheduler system use. The `bacct`(1) command produces one form of summary. The `lsb.acct` file is a text file suitable for processing with `awk`, `perl`, or similar tools. See the `lsb.acct`(5) online help page for details of the contents of this file. Additionally, the LSF API supports calls to process the `lsb.acct` records. See the *LSF Programmer's Guide* for details of LSF API.

If the `lsb.acct` file grows too large, you can move the `lsb.acct` file to a backup location. Removing `lsb.acct` will not cause any operational problems for LSF JobScheduler. The daemon automatically creates a new `lsb.acct` file to replace the moved file.

LSF JobScheduler Event Log

The LSF JobScheduler daemons keep an event log in the `lsb.events` file. The `mbatchd` daemon uses this information to recover from server failures, host reboots, and LSF JobScheduler reconfiguration. The `lsb.events` file is also used by the `bhist` command to display detailed information about the execution history of jobs, and by the `badmin` command to display the operational history of hosts, queues and LSF JobScheduler daemons.

For performance reasons, the `mbatchd` automatically backs up and rewrites the `lsb.events` file after every 1000 job completions (this is the default; the value is controlled by the `MAX_JOB_NUM` parameter in the `lsb.param` file). The old `lsb.events` file is moved to `lsb.events.1`, and each old `lsb.events.`*n* file is moved to `lsb.events.`*n+1*. The `mbatchd` never deletes these files. If disk storage is a concern, the LSF administrator should arrange to archive or remove old `lsb.events.`*n* files occasionally.

CAUTION!

Do not remove or modify the `lsb.events` file. Removing or modifying the `lsb.events` file could cause jobs to be lost.

Duplicate Event Logging

By default, LSF JobScheduler stores all state information needed to recover from server failures, host reboots, or reconfiguration in a file in the LSB_SHAREDIR directory. Typically, the LSB_SHAREDIR directory resides on a reliable file server which also contains other critical applications necessary for running users' jobs.This is done because, if the central file server is unavailable, users' applications cannot run, and the failure of LSF JobScheduler to continue processing users' jobs is a secondary issue.

For sites not wishing to rely solely on a central file server for recovery information, LSF can be configured to maintain a replica of the recovery file. The replica is stored on the file server, and used if the primary copy is unavailable—referred to as *duplicate event logging*. When LSF is configured this way, the primary even log is stored on the first master host, and re-synchronized with the replicated copy when the host recovers.

Configuring Duplicate Event Logging

To enable the replication feature, define LSB_LOCALDIR in the `lsf.conf` file. LSB_LOCALDIR should be a local directory and it should exist ONLY on the first master host (i.e. the first host configured in the `lsf.cluster.`*`cluster`* file).

LSB_LOCALDIR is used to store the primary copy of the batch state information. The contents of LSB_LOCALDIR are copied to a replica in LSB_SHAREDIR which resides on a central file server. As before, LSB_SHAREDIR is assumed to be accessible from all hosts which can potentially become the master.

How Duplicate Event Logging Works

With the replication feature enabled the following scenarios can occur.

Failure of File Server

If the file server containing LSB_SHAREDIR goes down, LSF will continue to process jobs. Client commands such as `bhist(1)` and `bacct(1)` which directly read LSB_SHAREDIR will not work. When the file server recovers, the replica in LSB_SHAREDIR will be updated.

Failure of First Master Host

If the first master host fails, then the primary copy of the recovery file in the LSB_LOCALDIR directory becomes unavailable. A new master host will be selected which will use the recovery file replica in LSB_SHAREDIR to restore its state and to log future events. There is no replication by the second master.

Recovery of First Master Host

When the first master host becomes available again, it will update the primary copy in LSB_LOCALDIR from the replica in LSB_SHAREDIR and continue operations as before.

The replication feature improves the reliability of LSF JobScheduler operations provided that the following assumptions hold:

- Failure of the LSF master host only occurs from the first master to the second master. The replication feature is not active if the second master also fails and a third master takes over.
- The master host containing LSB_LOCALDIR and the file server containing LSB_SHAREDIR do not fail simultaneously. In this situation, LSF JobScheduler will be unavailable.
- Network partitioning causing a cluster to split into two independent clusters each simultaneously running an `mbatchd` does not occur. This may happen given certain network topologies and failure modes. For example, connectivity is lost between the first master, M1, and both the file server and the secondary master, M2. Both M1 and M2 will run the `mbatchd` service with M1 logging events to LSB_LOCALDIR and M2 logging to LSB_SHAREDIR. When connectivity is restored, the changes made by M2 to LSB_SHAREDIR will be lost when M1 updates LSB_SHAREDIR from its copy in LSB_LOCALDIR.

Controlling LSF JobScheduler Servers

The `lsadmin` command is used to control LSF Base daemons, LIM and RES. LSF JobScheduler has the `badmin` command to perform similar operations on LSF JobScheduler daemons.

LSF JobScheduler System Status

To check the status of LSF JobScheduler server hosts and queues, use the `bhosts` and `bqueues` commands:

```
% bhosts
HOST_NAME   STATUS   MAX  NJOBS  RUN  SSUSP USUSP  RSV
hostA       ok       1    0      0    0     0      0
hostB       closed   2    2      2    0     0      0
hostD       ok       8    1      1    0     0      0


% bqueues
QUEUE_NAME     PRIO   STATUS          NJOBS  PEND  RUN  SUSP
night          30     Open:Inactive   4      4     0    0
short          10     Open:Active     1      0     1    0
simulation     10     Open:Active     0      0     0    0
default        1      Open:Active     6      4     2    0
```

If the status of a server is ‘closed’, then it will not accept more jobs. A server host can become closed if one of the following conditions is true:

- The host reaches its job slot limit

- The LSF administrator has explicitly closed the server host using the `badmin` command

An inactive queue will accept new job submissions, but will not dispatch any new jobs. A queue can become inactive if the LSF cluster administrator explicitly inactivates it via `badmin` command.

`mbatchd` automatically logs the history of the LSF JobScheduler daemons in the LSF JobScheduler event log. You can display the administrative history of the batch system using the `badmin` command.

The `badmin hhist` command displays the times when LSF JobScheduler server hosts are opened and closed by the LSF administrator.

The `badmin qhist` command displays the times when queues are opened, closed, activated, and inactivated.

The `badmin mbdhist` command displays the history of the `mbatchd` daemon, including the times when the master starts, exits, reconfigures, or changes to a different host.

The `badmin hist` command displays all LSF JobScheduler history information, including all the events listed above.

Remote Start-up of `sbatchd`

You can use `badmin hstartup` command to start `sbatchd` on some or all remote hosts from one host:

```
% badmin hstartup all
Start up slave batch daemon on <hostA> ......done
Start up slave batch daemon on <hostB> ......done
Start up slave batch daemon on <hostD> ......done
```

UNIX For remote startup to work, file `/etc/lsf.sudoers` has to be set up properly and you have to be able to run `rsh` across all LSF hosts without having to enter a password. See *'The lsf.sudoers File'* on page 89 for configuration details of `lsf.sudoers`.

Restarting `sbatchd`

`mbatchd` is restarted by the `badmin reconfig` command. `sbatchd` can be restarted using the `badmin hrestart` command:

```
% badmin hrestart hostD
Restart slave batch daemon on <hostD> ...... done
```

You can specify more than one host name to restart `sbatchd` on multiple hosts, or use '`all`' to refer to all LSF JobScheduler server hosts. Restarting `sbatchd` on a host does not affect jobs that are running on that host.

Shutting Down LSF JobScheduler Daemons

The `badmin hshutdown` command shuts down the `sbatchd`.

```
% badmin hshutdown hostD
Shut down slave batch daemon on <hostD> .... done
```

If `sbatchd` is shut down, that particular host will not be available for running new jobs. Existing jobs running on that host will continue to completion, but the results will not be sent to the user until `sbatchd` is later restarted.

To shut down `mbatchd`, you must first use the `badmin hshutdown` command to shut down the `sbatchd` on the master host, and then run the `badmin reconfig` command. The `mbatchd` is normally restarted by `sbatchd`; if there is no `sbatchd` running on the master host, `badmin reconfig` causes `mbatchd` to exit.

If `mbatchd` is shut down, all LSF JobScheduler service will be temporarily unavailable. However all existing jobs will not be affected. When `mbatchd` is later restarted, previous status will be restored from the event log file and job scheduling will continue.

Opening and Closing of Server Hosts

Occasionally you may want to drain a server host for purposes of rebooting, maintenance, or host removal. This can be achieved by running the `badmin hclose` command:

```
% badmin hclose hostB
Close <hostB> ...... done
```

When a host is open, LSF JobScheduler can dispatch jobs to it. When a host is closed no new batch jobs are dispatched, but jobs already dispatched to the host continue to execute. To reopen a server host, run `badmin hopen` command:

```
% badmin hopen hostB
Open <hostB> ...... done
```

To view the history of a JobScheduler server host, run `badmin hhist` command:

```
% badmin hhist hostB
Wed Nov 20 14:41:58: Host <hostB> closed by administrator <lsf>.
Wed Nov 20 15:23:39: Host <hostB> opened by administrator <lsf>.
```

Controlling LSF JobScheduler Queues

Each JobScheduler queue can be open or closed, active or inactive. Users can submit jobs to open queues, but not to closed queues. Active queues start jobs on available server hosts, and inactive queues hold all jobs. The LSF administrator can change the state of any queue.

bqueues — Queue Status

The current status of a particular queue or all queues is displayed by the `bqueues(1)` command. The `bqueues -l` option also gives current statistics about the jobs in a particular queue such as the total number of jobs in this queue, the number of jobs running, suspended, etc.

```
% bqueues normal
QUEUE_NAME     PRIO     STATUS        NJOBS  PEND  RUN  SUSP
normal         30       Open:Active   6      4     2    0
```

Opening and Closing Queues

When a queue is open, users can submit jobs to the queue. When a queue is closed, users cannot submit jobs to the queue. If a user tries to submit a job to a closed queue, an error message is printed and the job is rejected. If a queue is closed but still active, previously submitted jobs continue to be processed. This allows the LSF administrator to drain a queue.

```
% badmin qclose normal
Queue <normal> is closed


% bqueues normal
QUEUE_NAME  PRIO     STATUS         NJOBS  PEND  RUN  SUSP
normal      30       Closed:Active  6      4     2    0

% bsub -q normal hostname
normal: Queue has been closed

% badmin qopen normal
Queue <normal> is opened
```

Activating and Inactivating Queues

When a queue is active, jobs in the queue are started if appropriate hosts are available. When a queue is inactive, jobs in the queue are not started. Queues can be activated and inactivated by the LSF administrator using `badmin qact` and `badmin qinact`.

If a queue is open and inactive, users can submit jobs to this queue but no new jobs are dispatched to hosts. Currently running jobs continue to execute. This allows the LSF administrator to let running jobs complete before removing queues or making other major changes.

```
% badmin qinact normal
Queue <normal> is inactivated

% bqueues normal
QUEUE_NAME     PRIO     STATUS         NJOBS  PEND  RUN  SUSP
normal         30       Open:Inactive  0      0     0    0

% badmin qact normal
Queue <normal> is activated
```

Managing LSF JobScheduler Configuration

The LSF JobScheduler cluster is a subset of the LSF Base cluster. All servers used by LSF JobScheduler must belong to the base cluster. However not all servers in the base cluster must provide LSF JobScheduler services.

LSF JobScheduler configuration consists of five files: `lsb.params`, `lsb.hosts`, `lsb.queues`, `lsb.alarms`, and `lsb.calendars`. These files are stored in `LSB_CONFDIR/`*cluster*`/configdir`, where *cluster* is the name of your cluster.

All these files are optional. If any of these files does not exist, LSF JobScheduler will assume a default configuration.

The `lsb.params` file defines general parameters about LSF JobScheduler system operation such as the name of the default queue when the user does not specify one, scheduling intervals for `mbatchd` and `sbatchd`, etc. Detailed parameters are described in *'The lsb.params File'* on page 93.

The `lsb.hosts` file defines LSF JobScheduler server hosts together with their attributes. Not all LSF hosts defined by LIM configuration have to be configured to run jobs. Server host attributes include scheduling load thresholds, job slot limits, etc. This file is also used to define host groups. See *'Host Section'* on page 95 for details of this file.

The `lsb.queues` file defines job queues. See *'The lsb.queues File'* on page 97 for more details.

The `lsb.alarms` file contains definitions of alarms used by LSF JobScheduler. See *'The lsb.alarms File'* on page 101 for details. The `lsb.calendars` file defines system calendars for usage by all user jobs. See *'The lsb.calendars File'* on page 103 for details.

When you first install LSF on your cluster, some example queues are already configured for you. You should customize these queues or define new queues to meet your site need.

Note

After changing any of the LSF JobScheduler configuration files, you need to run `badmin reconfig` *to tell* `mbatchd` *to pick up the new configuration. You also must run this every time you change LIM configuration.*

Adding a JobScheduler Server Host

To add a server host to an LSF JobScheduler configuration, use the following procedure.

Step 1 If you are adding a host that has not been added to the LSF Base cluster yet, do the steps described in '*Adding a Host to a Cluster*' on page 32.

Step 2 Modify the `LSB_CONFDIR/`*cluster*`/configdir/lsb.hosts` file to add the new host together with its attributes. If you want to limit the added host for use only by some queues, you should also update the `lsb.queues` file. Since host types and host models, as well as the virtual name '`default`', can be used to refer to all hosts of that type, model, or every other LSF host not covered by the definitions, you may not need to change any of the files, if the host is already covered.

Step 3 Run `badmin reconfig` to tell `mbatchd` to pick up the new configuration.

Step 4 Start `sbatchd` on the added host by running `badmin hstartup` or simply start it by hand.

Removing a JobScheduler Server Host

To make a host no longer a server host, use the following procedure.

Step 1 If you need to permanently remove a host from your cluster, you should use `badmin hclose` to prevent new batch jobs from starting on the host, and wait for any running jobs on that host to finish. If you wish to shut the host down before all jobs complete, use `bkill` to kill the running jobs.

Step 2 Modify `lsb.hosts` and `lsb.queues` in `LSB_CONFDIR`/*cluster*/`configdir` directory and remove the host from any of the sections.

Step 3 Run `badmin hshutdown` to shutdown `sbatchd` on that host.

CAUTION!

You should never remove the master host from LSF JobScheduler. Change the LIM configuration to assign a different default master host if you want

to remove your current default master from the LSF JobScheduler server pool.

Adding a Queue

To add a queue to a cluster, use the following procedure.

Step 1 Log in as the LSF administrator on any host in the cluster.

Step 2 Edit the `LSB_CONFDIR/cluster/configdir/lsb.queues` file to add the new queue definition. You can copy another queue definition from this file as a starting point; remember to change the `QUEUE_NAME` of the copied queue. Save the changes to `lsb.queues`. See *'The lsb.queues File'* on page 97 for a complete description of LSF JobScheduler queue configuration.

Step 3 Run the command `badmin ckconfig` to check the new queue definition. If any errors are reported, fix the problem and check the configuration again. See *'Controlling LSF JobScheduler Servers'* on page 55 for an example of normal output from `badmin ckconfig`.

Step 4 When the configuration files are ready, run `badmin reconfig`. The master batch daemon (`mbatchd`) is unavailable for approximately one minute while it reconfigures. Pending and running jobs are not affected.

Adding a queue does not affect pending or running LSF JobScheduler jobs.

Removing a Queue

Before removing a queue, you should make sure there are no jobs in that queue. If you remove a queue that has jobs in it, the jobs are temporarily moved to a *lost and found* queue. Jobs in the lost and found queue remain pending until the user or the LSF administrator uses the `bswitch` command to switch the jobs into regular queues. Jobs in other queues are not affected.

In this example, move all pending and running jobs in the *night* queue to the *idle* queue, and then delete the *night* queue.

Step 1 Log in as the LSF administrator on any host in the cluster.

Step 2 Close the queue to prevent any new jobs from being submitted:

```
% badmin qclose night
Queue <night> is closed
```

Step 3 Move all pending and running jobs into another queue. The `bswitch -q night` argument chooses jobs from the *night* queue, and the job ID number `0` specifies that all jobs should be switched.

```
% bjobs -u all -q night
JOBID USER   STAT QUEUE  FROM_HOST EXEC_HOST JOB_NAME   SUBMIT_TIME
5308  user5  RUN  night  hostA     hostD     sleep 500  Nov 21 18:16
5310  user5  PEND night  hostA               sleep 500  Nov 21 18:17
```

```
% bswitch -q night idle 0
Job <5308> is switched to queue <idle>
Job <5310> is switched to queue <idle>
```

Step 4 Edit the `LSB_CONFDIR/`*`cluster`*`/configdir/lsb.queues` file. Remove (or comment out) the definition for the queue being removed. Save the changes.

Step 5 Run the command `badmin reconfig`. If any problems are reported, fix them and run `badmin reconfig` again. The JobScheduler system is unavailable for about one minute while the system rereads the configuration.

Controlling LSF JobScheduler Jobs

The LSF administrator can control jobs belonging to any user. Other users may control only their own jobs. Jobs can be suspended, resumed and killed.

The `bstop`, `bresume`, `bkill`, and `bdel` commands send signals to batch jobs. See the `kill(1)` man / online help page for a discussion of these signals.

`bstop` suspends a job, bringing the suspended job to USUSP status.

`bresume` causes a suspended job to resume execution.

`bkill` terminates a job.

`bdel` deletes a job.

See the *LSF JobScheduler User's Guide* and the online help pages for more information about these commands.

This example shows the use of the `bstop` and `bkill` commands:

```
% bstop 5310
Job <5310> is being stopped

% bjobs 5310
JOBID USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
5310  user5 PSUSP night  hostA                 analysis  Nov 21 18:17

% bkill 5310
Job <5310> is being terminated

% bjobs 5310
JOBID USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
5310  user5 EXIT  night  hostA                 analysis  Nov 21 18:17
```

Controlling Job Execution Environment

When LSF JobScheduler runs your jobs, it tries to make it as transparent to the user as possible. By default, the execution environment is maintained to be as close to the current environment as possible. LSF JobScheduler will copy the environment from the submission host to the execution host. It also sets the `umask` and the current working directory.

Since a network can be heterogeneous, it is often impossible or undesirable to reproduce the submission host's execution environment on the execution host. For example, if a home directory is not shared between submission and execution host, LSF JobScheduler runs the job in `/tmp` on the execution host.

Users can change the default behaviour by using a job starter. See *'Using a Job Starter'* on page 66 for details of a job starter.

In addition to environment variables inherited from the user, LSF JobScheduler also sets a few more environment variables for jobs. These are:

- `LSB_JOBID`: Job ID assigned by LSF JobScheduler.
- `LSB_HOSTS`: The list of hosts that are used to run the job. For sequential jobs, this is only one host name. For parallel jobs, this includes multiple host names.
- `LSB_QUEUE`: The name of the queue the job belongs to.
- `LSB_JOBNAME`: Name of the job.
- `LSB_EXIT_PRE_ABORT`: Set to an integer value representing an exit status. A pre-execution command should exit with this value if it wants the job to be aborted instead of requeued or executed.
- `LSB_JOB_STARTER`: Set to the value of the job starter if a job starter is defined for the queue.
- `LSB_EVENT_ATTRIB`: Set to the attributes of external events that were specified in the job's dependency condition. The variable is of the format '`event_name1 attribute1 event_name2 ...`'.
- `LS_JOBPID`: Set to the process ID of the job.
- `LS_SUBCWD`: This is the directory on the submission when the job was submitted. This is different from `PWD` only if the directory is not shared across machines or when the execution account is different from the submission account as a result of account mapping.

Note

These variables are set for the convenience of the job. The job does not have to use these variables.

Using a Job Starter

Some jobs have to be started under particular shells or require certain setup steps to be performed before the actual job is executed. This is often handled by writing wrapper scripts around the job. The LSF job starter feature allows you to specify an executable which will perform the actual execution of the job, doing any necessary setup before hand.The job starter can be specified at the queue level using the `JOB_STARTER` parameter in the `lsb.queues` file. This allows the LSF JobScheduler queue to control the job startup. For example, the following might be defined in a queue:

```
Begin Queue
.
JOB_STARTER = xterm -e
.
End Queue
```

This way all jobs submitted into this queue will be run under an xterm.

The following are other possible uses of a job starter:

- Set job starter to '`/bin/csh -c`' allows C-shell syntax to be used.
- Set job starter to '`$USER_STARTER`' enables users to define his / her own job starter by defining the environment variable `USER_STARTER`.

A job starter is configured at the queue level. See '*Job Starter*' on page 101 for details.

System Calendars

Calendars are normally created by users using the `bcadd` command or the `xbcal` GUI interface. Calendars that are commonly used may be defined as system calendars, which can be referenced by all users. System calendars are defined in the `lsb.calendars` configuration file in `LSB_CONFDIR/`*cluster*`/configdir` directory.

The `lsb.calendars` file consists of multiple `Calendar` section where each section corresponds to one calendar. Each calendar section requires the `NAME` and

TIME_EVENTS parameter and can optionally contain a DESCRIPTION parameter. The Calendar section is of the form:

```
Begin Calendar
NAME=<name>
CAL_EXPR=<calendar expression>
DESCRIPTION=<description>
End Calendar
```

The syntax of the CAL_EXPR parameter is described in the *LSF JobScheduler User's Guide* and the man page bcaladd(1). The following is a sample lsb.calendars file:

```
Begin Calendar
NAME=Holidays
CAL_EXPR=((*:Dec:25)||(*:Jan:1)||*:Jul:4))
DESCRIPTION=U.S. Holidays
End Calendar

Begin Calendar
NAME=weekends
CAL_EXPR=sat,sun
DESCRIPTION=weekend days
End Calendar
```

System calendars are owned by the virtual user sys and can be viewed by everybody. The xbcal GUI and the bcal command display the system calendars:

```
% bcal
CALENDAR_NAME   OWNER   STATUS     LAST_CAL_DAY       NEXT_CAL_DAY
holiday         sys     inactive   Fri Jul 4 1997     Thu Dec 25 1997
weekend         sys     inactive   Sun Dec 21 1997    Sat Dec 27 1997
workday         sys     active     Tue Dec 23 1997    Wed Dec 24 1997
```

System calendars cannot be created with the bcadd command and they cannot be deleted with the bcdel command. When a system calendar is defined, its name becomes a reserved calendar name in the cluster. Consequently, users cannot create a calendar with the same name as a system calendar.

External Event Management

LSF JobScheduler supports the scheduling of jobs based on external site-specific events. A typical use of this feature in data processing environment is to trigger jobs based on the arrival of data or the availability of tapes. Sites that use storage management systems, for example, can coordinate the dispatch of jobs with the staging of data from hierarchical storage onto disk.

The scheduling daemon (`mbatchd`) can startup and communicate with an external event daemon (`eeventd`) to detect the occurrence of events. The `eeventd` is implemented as an executable called `eeventd` which resides in `LSF_SERVERDIR`. Users can submit jobs specifying dependencies on any logical combination of external events using the `-w` option of the `bsub` command. External event dependencies can be combined with job, file, and calendar events.

A protocol is defined which allows `mbatchd` to indicate to the `eeventd` that a job is waiting on a particular event. The `eeventd` will monitor the event and possibly take actions to trigger it. When the event occurs, the `eeventd` informs `mbatchd`, which will then consider the job as eligible for dispatch provided appropriate hosts are available.

LSF JobScheduler comes with an `eeventd` for file event detection. If you want to monitor additional site events, you can simply add event detection functions into the existing `eeventd`. The source code of the default `eeventd` is also included in the release.

The `eeventd` Protocol

The protocol between the external event daemon, `eeventd`, and `mbatchd` consists of a sequence of ASCII messages that are exchanged over a socket pair.

UNIX The startup sequence and message format for the protocol is described in the man page `eeventd`(8).

Each event is identified by an *event name*. The event name is an arbitrary string, which is site-specific. A user specifies job dependencies on an external event by using the `-w` option of the `bsub` command using the `event` keyword. For example:

```
% bsub -w 'event(tapeXYZ)' myjob
```

LSF JobScheduler considers the job to be waiting on an event with the name `tapeXYZ`. There is no checking of the syntax of the event name by LSF JobScheduler. The `eeventd` can reject an event if the syntax is incorrect preventing the job from being dispatched until the user either modifies the event or removes the job. Alternatively, a site may write a wrapper submission script which checks the syntax of the event before it is submitted to LSF JobScheduler.

The following messages are sent from `mbatchd` to the `eeventd`:

`SUB event_name`

Subscribe to the event given by *event_name*. Whenever a job is submitted with a new event name that `mbatchd` has not seen before, a subscribe request is sent to the `eeventd`. The `eeventd` is expected to monitor the event, and if necessary, to take any actions required for the event to occur.

`UNSUB event_name`

Unsubscribe to a given event when there are no jobs dependent on this event. This should cause the `eeventd` to stop monitoring the event.

The following messages are sent from the `eeventd` to `mbatchd`:

`START event_name event_type [event_attrib]`

Tells `mbatchd` to make the event active. The *event_type* field should be one of `latched`, `pulse`, `pulseAll`, or `exclusive`. The different event types control when `mbatchd` will inactivate an event as follows:

`latched`	Not automatically inactivated until an explicit `END` message is received.
`pulse`	Automatically inactivated when one job is dispatched. Subsequent `START` messages on the same event can cause one job to be dispatched, each time the event is pulsed.
`pulseAll`	Automatically inactivated after it is received. For `pulseAll` events, each job will maintain its own copy of the event state. When a `pulseAll` event is triggered, all jobs currently waiting on the event will have their copy of the event state marked as active and will be eligible for dispatch. Subsequently submitted jobs will view the event as inactive.

`exclusive` Automatically inactivated when one job is dispatched and kept inactive until the job completes. Subsequent attempts by the `eeventd` to activate the event are ignored until job completion.

The *event_attrib* is an optional attribute string that can be associated with the event. The event attribute is not interpreted by the system and is passed to a job when it starts via the `LSB_EVENT_ATTRIB` environment variable. It can be used to communicate information between the event daemon and the job. It is also displayed by the `bevents` command.

`END event_name`

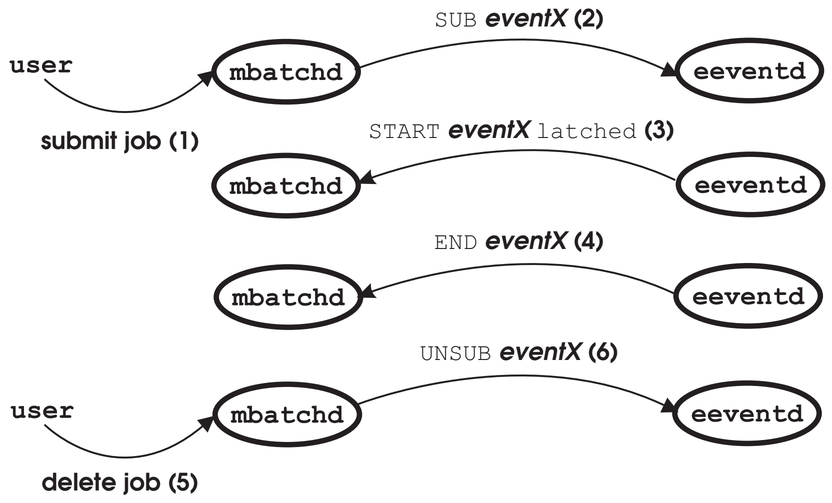
Causes the event to be put in the inactive state. If the event is already inactive, this has no effect.

`REJECT event_name [event_attribute]`

Causes the event to be put in the reject state. This can be used to indicate a syntax error in the event name. Rejected events are considered to be inactive so that jobs waiting on them are not dispatched. The optional *event_attrib* can be used to give more information about why the job is rejected. This information will be displayed by the `bevents` command.

The sequence of interactions between `mbatchd` and the `eeventd` are shown in *Figure 2*.

Figure 2. `mbatchd` and `eeventd` Interactions



Step 1 User submits a job specifying dependency on the external event *eventX*

Step 2 `mbatchd` scans event table to see if *eventX* already exists. If not, it creates the event and sends a subscribe message to the `eeventd`. The `eeventd` recognizes *eventX* and initiates monitoring it. If the `eeventd` could not recognize *eventX*, it returns a `REJECT` message to `mbatchd`.

Step 3 The `eeventd` detects an occurrence of *eventX* and sends a `START` message telling `mbatchd` to try to schedule any jobs waiting on the event. Since *eventX* is `latched`, `mbatchd` will consider the event as active indefinitely. If a job also has a dependency on a calendar, it can be run multiple times while *eventX* is still active.

Step 4 The `eeventd` detects that *eventX* is no longer occurring and sends an `END` message. `mbatchd` considers *eventX* as inactive and stops scheduling jobs waiting on the event.

Steps (*3*) and (*4*) can be repeated multiple times.

Step 5 The user deletes the job waiting on *eventX*.

Step 6 If there are no jobs in the system waiting on *eventX*, an `UNSUB` message is sent to the `eeventd`. This should cause the `eeventd` to stop monitoring *eventX*.

The external event daemon given in `examples/eevent/eevent.c` provides an example of a simple event daemon. It receives requests from `mbatchd` to subscribe and unsubscribe to events. Periodically, it scans the list of subscribed events and toggles the state of the event between active and inactive. The type of the event is chosen based on the event name, that is event names beginning with the string 'exclusive' or 'pulse' are treated as `exclusive` events or `pulse` events respectively. Otherwise the event is treated as a `latched` event.

File Event Handling

The handling of file events is implemented using the default external event daemon. The installation scripts automatically install the event daemon that handles file events.

Since only one external event daemon can run on a system, sites requiring file event handling in addition to site-specific events must modify the existing file event daemon. The source is provided in `examples/eevent/fevent.c` in the distribution directory.

You can monitor all external events using `bevents` command.

Alarm Configuration

The LSF JobScheduler system can raise an alarm when exceptions are encountered in processing critical jobs. See the *LSF JobScheduler User's Guide* to see how users can associate an alarm with a failure in a job. Alarms must be configured by the LSF JobScheduler administrator before they can be associated with jobs by users. The alarm configuration defines how a notification is to be delivered.

The administrator can configure an alarm to send notification via email or invoke a command which sends a page or a message to a system console. Each time an alarm is triggered a record is created for the incident in a log. The administrator or other users with write access to the log can modify the state of the incident using the `xbalarms` GUI or the `balarms` command.

The alarm system supplied with LSF JobScheduler consists of the following components:

- a configuration file, `lsb.alarms`, in LSB_CONFDIR`/cluster/configdir` which defines the alarms
- a program, `raisealarm`, in LSF_SERVERDIR, which is invoked by the `mbatchd` in order to trigger an alarm
- a log file, `lsb.alarmlog`, in LSB_SHAREDIR`/cluster/logdir` which contains a record of each alarm incident (a history log file, `lsb.alarmlog.hist`, keeps old alarm incident records)
- a daemon, `alarmd`, started by `mbatchd` performs periodic operations on the alarm
- logging, such as sending renotifications or moving resolved or expired records to the history log
- GUI and command line tools for viewing, acknowledging and resolving alarms

Alarm definitions must be created by the LSF Administrator in the `lsb.alarms` file located in LSB_CONFDIR`/cluster/configdir` directory. The `lsb.alarms` file consists of multiple “Alarm” sections where each section corresponds to one alarm. Each section has a number of keywords which are used to define the alarm parameters. Each alarm section requires the NAME and NOTIFICATION parameters. See *‘The lsb.alarms File’* on page 101 for detailed information on each parameter.

4. LSF Base Configuration Reference

This chapter contains a detailed description of the contents of the LSF Base configuration files. These include the installation file `lsf.conf`; the LIM configuration files `lsf.shared`, `lsf.cluster.`*`cluster,`* and the optional LSF hosts file for additional host name information.

The `lsf.conf` File

Installation of and operation of LSF is controlled by the `lsf.conf` file. The `lsf.conf` file is created during installation, and records all the settings chosen when LSF is installed. This information is used by LSF daemons and commands to locate other configuration files, executables, and network services.

`lsf.conf` contains LSF installation settings as well as some system wide options. This file is initially created by the `lsfsetup` utility during LSF installation, and updated if necessary when you upgrade to a new version. Many of the parameters are set during the installation. This file can also be expanded to include LSF application specific parameters.

LSB_CONFDIR

LSF JobScheduler configuration directories are installed under `LSB_CONFDIR`. Configuration files for each LSF cluster are stored in a subdirectory of `LSB_CONFDIR`. This subdirectory contains several files that define the LSF JobScheduler user and host lists, operation parameters, and queues.

All files and directories under `LSB_CONFDIR` must be readable from all hosts in the cluster. `LSB_CONFDIR/`*`cluster/`*`configdir` must be owned by the LSF administrator.

Default: `LSF_CONFDIR/lsbatch`

You should not try to redefine this parameter once LSF has been installed.

LSB_DEBUG

If this is defined, LSF JobScheduler will run in single user mode. In this mode, no security checking is performed, so the LSF JobScheduler daemons should not run as `root`. When LSB_DEBUG is defined, LSF JobScheduler will not look in the system services database for port numbers. Instead, it uses port number 40000 for `mbatchd` and port number 40001 for `sbatchd` unless `LSB_MBD_PORT` / `LSB_SBD_PORT` are defined in the file `lsf.conf`. The valid values for `LSB_DEBUG` are 1 and 2. You should always choose 1 unless you are testing LSF JobScheduler.

Default: undefined

LSB_MAILPROG

LSF JobScheduler normally uses `/usr/lib/sendmail` as the mail transport agent to send mail to users. If your site does not use `sendmail`, configure `LSB_MAILPROG` with the name of a `sendmail`-compatible transport program. LSF JobScheduler calls `LSB_MAILPROG` with the following arguments:

```
LSB_MAILPROG -F "LSF  system" -f Manager@host dest_addr
```

The `-F "LSF System"` argument sets the full name of the sender; the `-f` *Manager@host* argument gives the return address for LSF JobScheduler mail, which is the LSF administrator's mailbox. *dest_addr* is the destination address, generated by the rules given for `LSB_MAILTO` above.

`LSB_MAILPROG` must read the body of the mail message from the standard input. The end of the message is marked by end-of-file. Any program or shell script that accepts the arguments and input and delivers the mail correctly can be used. `LSB_MAILPROG` must be executable by any user.

If this parameter is modified, the LSF administrator must restart the `sbatchd` daemons on all hosts to pick up the new value.

Default: `/usr/lib/sendmail`

LSB_MAILTO

LSF JobScheduler sends electronic mail to users when their jobs complete or have errors, and to the LSF administrator in the case of critical errors in the LSF JobScheduler system. The default is to send mail to the user who submitted the job, on the host where the daemon is running; this assumes that your electronic mail system forwards messages to a central mailbox.

The `LSB_MAILTO` parameter changes the mailing address used by LSF JobScheduler. `LSB_MAILTO` is a format string that is used to build the mailing address. The substring `!U`, if found, is replaced with the user's account name; the substring `!H` is replaced with the name of the submission host. All other characters (including any other '`!`') are copied exactly. Common formats are:

`!U` - mail is sent to the submitting user's account name on the local host

`!U@!H` - mail is sent to `user@submission_hostname`

`!U@company_name.com` - mail is sent to `user@company_name.com`

If this parameter is modified, the LSF administrator must restart the `sbatchd` daemons on all hosts to pick up the new value.

Default: `!U`

LSB_SHAREDIR

LSF JobScheduler keeps job history and accounting log files for each cluster. These files are necessary for correct operation of the system. Like the organization under `LSB_CONFDIR`, there is one subdirectory for each cluster.

The `LSB_SHAREDIR/`*`cluster`*`/logdir` directory must be owned by the LSF administrator.

Default: `LSF_INDEP/work`

Note

All files and directories under LSB_SHAREDIR *must allow read and write access from the LSF master host. See 'Fault Tolerance' on page 2 and 'Using LSF JobScheduler without Shared File Systems' on page 5.*

LSF_BINDIR

Directory where all user commands are installed.

Default: `LSF_MACHDEP/bin`

LSF_CONFDIR

The directory where all LIM configuration files are installed. These files are shared throughout the system and should be readable from any host. This directory can contain configuration files for more than one cluster.

Default: `LSF_INDEP/conf`

LSF_ENVDIR

LSF normally installs the `lsf.conf` file in the `/etc` directory. The `lsf.conf` file is installed by creating a shared copy in `LSF_SERVERDIR` and adding a symbolic link from `/etc/lsf.conf` to the shared copy. If `LSF_ENVDIR` is set, the symbolic link is installed in `LSF_ENVDIR/lsf.conf`.

Default: `/etc`

LSF_INCLUDEDIR

Directory under which the LSF API header file `<lsf/lsf.h>` is installed.

Default: `LSF_INDEP/include`

LSF_INDEP

Specifies the default top-level directory for all host type independent LSF files. This includes manual pages, configuration files, working directories, and examples. For example, defining `LSF_INDEP` as `/usr/local/lsf/mnt` places manual pages in /

`usr/local/lsf/mnt/man`, configuration files in `/usr/local/lsf/mnt/conf`, and so on.

Default: `/usr/local/lsf/mnt`

LSF_LIBDIR

Directory where the LSF application programming interface library `liblsf.a` is installed.

Default: `LSF_MACHDEP/lib`

LSF_LICENSE_FILE

The full path name of the FLEXlm license file used by LSF. This variable is set to LSF_CONFDIR/`license.dat` by default at installation time.

UNIX Default: `/usr/local/flexlm/licenses/license.dat`

NT Default: `C:\Flexlm\License.dat`

LSF_LIM_DEBUG

If `LSF_LIM_DEBUG` is defined, the Load Information Manager (LIM) will operate in single user mode. No security checking is performed, so LIM should not run as root. LIM will not look in the services database for the LIM service port number. Instead, it uses port number 36000 unless `LSF_LIM_PORT` has been defined. The valid values for `LSF_LIM_DEBUG` are 1 and 2. You should always choose 1 unless you are testing LSF.

Default: undefined

LSF_LIM_PORT, LSF_RES_PORT, LSB_MBD_PORT, LSB_SBD_PORT

Internet port numbers are used for communication with the LSF daemons. The port numbers are normally obtained by looking up the LSF service names in the `/etc/`

`services` file or the services YP map. If it is not possible to modify the service database, these variables can be defined to set the port numbers.

With careful use of these settings along with the `LSF_ENVDIR` and `PATH` environment variables, it is possible to run two versions of the LSF software on a host, selecting between the versions by setting the `PATH` environment variable to include the correct version of the commands and the `LSF_ENVDIR` environment variable to point to the directory containing the appropriate `lsf.conf` file.

Default: get port numbers from services database

LSF_LOGDIR

This is an optional definition.

If `LSF_LOGDIR` is defined, error messages from all servers are logged into files in this directory. If a server is unable to write in this directory, then the error logs are created in `/tmp`.

UNIX If `LSF_LOGDIR` is not defined, then `syslog` is used to log everything to the system log using the `LOG_DAEMON` facility. The syslog facility is available by default on most UNIX systems. The `/etc/syslog.conf` file controls the way messages are logged, and the files they are logged to. See the manual pages for the `syslogd` daemon and the `syslog` function for more information.

UNIX Default: log messages go to `syslog`
Windows NT Default: log messages lost if LSF_LOGDIR undefined

LSF_LOG_MASK

The error message log level for LSF daemons. This definition applies no matter where the LSF daemons are logging messages. All messages logged at the specified level or higher are recorded; lower level messages are discarded. The log levels in order from highest to lowest are:

- `LOG_ALERT`
- `LOG_SALERT`

- `LOG_EMERG`
- `LOG_ERR`
- `LOG_CRIT`
- `LOG_WARNING`
- `LOG_NOTICE`
- `LOG_INFO`
- `LOG_DEBUG`

Most important LSF log messages are at the `LOG_ERR` or `LOG_WARNING` level. Messages at the `LOG_INFO` and `LOG_DEBUG` level are only useful for debugging.

Default: `LOG_WARNING`

LSF_MACHDEP

Specifies the directory where host type dependent files are installed. The machine dependent files are the user programs, daemons, and libraries.

Default: `/usr/local/lsf`

LSF_MANDIR

Directory under which all manual pages are installed. The manual pages are placed in the `man1`, `man3`, `man5` and `man8` subdirectories of the `LSF_MANDIR` directory.

Default: `LSF_INDEP/man`

LSF_MISC

Directory where miscellaneous machine independent files such as LSF example source programs and scripts are installed.

Default: `LSF_CONFDIR/misc`

LSF_RES_DEBUG

If `LSF_RES_DEBUG` is defined, the Remote Execution Server (RES) will operate in single user mode. No security checking is performed, so RES should not run as root. RES will not look in the services database for the RES service port number. Instead, it uses port number 36002 unless `LSF_RES_PORT` has been defined. The valid values for `LSF_RES_DEBUG` are 1 and 2. You should always choose 1 unless you are testing RES.

Default: undefined

LSF_SERVERDIR

Directory where all server binaries are installed. These include `lim`, `res`, `nios`, `sbatchd`, `mbatchd`, `eeventd`. If you use `elim`, `eauth`, `eexec`, `esub`, etc, they should also be installed in this directory.

Default: `LSF_MACHDEP/etc`

LSF_SERVER_HOSTS

This defines one or more LSF server hosts that the application should contact to get in touch with a Load Information Manager (LIM). This is used on client hosts where no LIM is running on the local host. The LSF server hosts are hosts that run LSF daemons and provide load sharing services. Client hosts are hosts that only run LSF commands or applications but do not provide services to any hosts.

If `LSF_SERVER_HOSTS` is not defined, the application tries to contact the LIM on the local host.

The host names in `LSF_SERVER_HOSTS` must be enclosed in quotes and separated by white space; for example:

```
LSF_SERVER_HOSTS="hostA hostD hostB"
```

Default: undefined

LSF_STRIP_DOMAIN

This is an optional definition.

If all the hosts in your cluster can be reached using short host names, you can configure LSF to use the short host names by specifying the portion of the domain name to remove. If your hosts are in more than one domain, or have more than one domain name, you can specify more than one domain suffix to remove, separated by a colon ':'.

For example, given this definition of `LSF_STRIP_DOMAIN`:

```
LSF_STRIP_DOMAIN=.foo.com:.bar.com
```

LSF accepts *hostA*, *hostA.foo.com*, and *hostA.bar.com* as names for host *hostA*, and uses the name *hostA* in all output. The leading period '.' is required.

Default: undefined

XLSF_APPDIR

The directory where X application default files for LSF products are installed. The LSF commands that use X look in this directory to find the application defaults. Users do not need to set environment variables to use the LSF X applications. The application default files are platform-independent.

Default: `LSF_INDEP/misc`

UNIX

XLSF_UIDDIR

The directory where Motif User Interface Definition files are stored. These files are platform specific.

Default: `LSF_LIBDIR/uid`

The `lsf.shared` File

The `lsf.shared` file contains definitions that are used by all load sharing clusters. This includes lists of cluster names, host types, host models, the special resources available, and external load indices.

Clusters

The mandatory “Cluster” section defines all cluster names recognized by the LSF system, with one line for each cluster.

The ClusterName keyword is mandatory. All cluster names referenced anywhere in the LSF system must be defined here. The file names of cluster-specific configuration files must end with the associated cluster name.

```
Begin Cluster
ClusterName
cluster1
cluster2
End Cluster
```

Host Types

The mandatory `HostType` section lists the valid host type names in the cluster. Each host is assigned a host type in the `lsf.cluster.`*`cluster`* file. All hosts that can run the same binary programs should have the same host type, even if they have different models of processor. LSF uses the host type as a default requirement for task placement. Unless specified otherwise, jobs are always run on hosts of the same type.

The TYPENAME keyword is mandatory. Host types are usually based on a combination of the hardware name and operating system. If a job does not have a resource requirement specified, LSF runs the job on a host of the same type as the submission host, so you should give careful consideration to the host type for each host in the cluster. If your site already has a system for naming host types, you can use the same names for LSF.

```
Begin HostType
TYPENAME
SUN41
NT86
ALPHA
HPPA
End HostType
```

Host Models

The mandatory `HostModel` section lists the various models of machines and gives the relative CPU speed for each model. LSF uses the relative CPU speed to normalize the CPU load indices so that jobs are more likely to be sent to faster hosts. The `MODELNAME` and `CPUFACTOR` keywords are mandatory.

It is up to you to identify the different host models in your system, but generally you need to identify first the distinct host types, such as HPPA and SPARC, and then the machine models within each, such as SparcIPC, Sparc1, Sparc2, and Sparc10.

Though it is not required, you would typically assign a CPU factor of 1.0 to the slowest machine model in your system, and higher numbers for the others. For example, for a machine model that executes at twice the speed of your slowest model, a factor of 2.0 should be assigned.

```
Begin HostModel
MODELNAME  CPUFACTOR
SparcIPC   1.0
Sparc10    2.0
End HostModel
```

The CPU factor affects the calculation of job execution time limits and accounting. Using large values for the CPU factor can cause confusing results when CPU time limits or accounting are used.

Resources

The optional “Resource” section contains a list of resource names. Resource names are character strings chosen by the LSF administrator. You can use any name other than the reserved resource names. The keywords `RESOURCENAME` and `DESCRIPTION` are mandatory.

For a more general discussion of boolean resources, see *Section 4, ‘Resources’, beginning on page 45* of the *LSF JobScheduler User's Guide.*

Resource names must be strings of numbers and letters, beginning with a letter and no more than 29 characters long. You can define up to 32 resource names in `lsf.shared`.

This sample `Resource` section defines boolean resources to represent processor types, operating system versions, and software licenses:

```
Begin Resource
RESOURCENAME  DESCRIPTION
sparc         (Sparc CPU)
sunos4        (Running SunOS 4.x)
solaris       (Running Solaris 2.x)
frame         (FrameMaker license)
End Resource
```

The `lsf.cluster.`*`cluster`* File

This is the load-sharing cluster configuration file. There is one such file for each load sharing cluster in the system. The *cluster* suffix must agree with the name defined in the `Cluster` section of the `lsf.shared` file.

Parameters

The `Parameters` section is optional. This section contains miscellaneous parameters for the LIM.

PRODUCTS

The `PRODUCTS` line specifies which LSF products will be enabled in the cluster. The `PRODUCT` can specify any combination of the strings 'LSF_Base', 'LSF_Batch', 'LSF_JobScheduler', 'LSF_MultiCluster', and 'LSF_Analyzer' to enable the operation of these products. If any of 'LSF_Batch', 'LSF_JobScheduler', or 'LSF_MultiCluster' are specified, then 'LSF_Base' is automatically enabled as well. Specifying the `PRODUCTS` line enables the product for all hosts in the cluster. Individual hosts can be configured to run as LSF JobScheduler servers or LSF Batch servers within the same cluster. LSF MultiCluster is either enabled or disabled for multicluster operation for the entire cluster.

The PRODUCTS line is created automatically by the installation program `lsfsetup`. For example:

```
Begin Parameters
PRODUCTS=LSF_Base LSF_JobScheduler
End Parameters
```

If the PRODUCTS line is not specified, the default is to enable the operation of LSF Base and LSF Batch.

Note

The products defined by the PRODUCTS *line must match the license file used to serve the cluster. A host will be unlicensed if the license is unavailable for the component it was configured to run. For example, if you configure a cluster to run LSF JobScheduler on all hosts, and the license file does not contain the LSF JobScheduler product, then the hosts will be unlicensed, even if there are licenses for LSF Base or LSF Batch.*

Default: `LSF_Base LSF_Batch`

LSF Administrators

The `ClusterAdmins` section defines the LSF administrator(s) for this cluster. Both UNIX user and group names may be specified with the `ADMINISTRATORS` keyword. The LIM will expand the definition of a group name using the `getgrnam(3)` call. The first administrator of the expanded list is considered the primary LSF administrator. The primary administrator is the owner of the LSF configuration files, as well as the working files under `LSB_SHAREDIR/`*`cluster`*. If the primary administrator is changed, make sure the owner of the configuration files and the files under `LSB_SHAREDIR/`*`cluster`* are changed as well. All LSF administrators have the same authority to perform actions on LSF daemons, jobs, queues, or hosts in the system.

For backwards compatibility, `ClusterManager` and `Manager` are synonyms for `ClusterAdmins` and `ADMINISTRATOR` respectively. It is possible to have both sections present in the same `lsf.cluster.`*`cluster`* file to allow daemons from different LSF versions to share the same file.

If this section is not present, the default LSF administrator is root. For flexibility, each cluster may have its own LSF administrator(s), identified by a username, although the same administrator(s) can be responsible for several clusters.

The ADMINISTRATOR parameter is normally set during the installation procedure.

Use the -l option of the lsclusters(1) command to display all the administrators within a cluster.

The following gives an example of a cluster with three LSF administrators. The user listed first, user2, is the primary administrator.

```
Begin ClusterAdmins
ADMINISTRATORS = user2 lsfgrp user7
End ClusterAdmins
```

Hosts

The Host section is the last section in lsf.cluster.*cluster* and is the only required section. It lists all the hosts in the cluster and gives configuration information for each host.

The order in which the hosts are listed in this section is important. The LIM on the first host listed becomes the master LIM if this host is up; otherwise, that on the second becomes the master if its host is up, and so on.

Since the master LIM makes all placement decisions for the cluster, you want it on a fast machine. Also, to avoid the delays involved in switching masters if the first machine goes down, you want the master to be on a reliable machine. It is desirable to arrange the list such that the first few hosts in the list are always in the same subnet. This avoids the situation where the second host takes over the master when there are communication problems between subnets.

Configuration information is of two types. Some fields in a host entry simply describe the machine and its configuration. Other fields set thresholds for various resources. Both types are listed below.

Descriptive Fields

The HOSTNAME, model, type, and RESOURCES fields must be defined in the Host section. The server, nd, RUNWINDOW and REXPRI fields are optional.

HOSTNAME - the official name of the host as returned by hostname(1). Must be listed in lsf.shared as belonging to this cluster.

`model` - host model. Must be one of those defined in the `lsf.shared` file. This determines the CPU speed scaling factor applied in load and placement calculations.

`type` - a host type as defined in the `HostType` section of `lsf.shared`. The strings used for host types are decided by the system administrator, e.g. SPARC, DEC, HPPA. The host type is used to identify binary-compatible hosts.

The host type is used as the default resource requirement. That is, if no resource requirement is specified in a placement request then the task is run on a host of the same type as the sending host.

Often one host type can be used for many machine models. For example, the host type name `SUN41` might be used for any computer with a SPARC processor running SunOS 4.1. This would include many Sun models and quite a few from other vendors as well.

`server` - 1 if the host can receive jobs from other hosts; 0 otherwise. If `server` is set to 0, the host is an LSF client. Client hosts do not run the LSF daemons. Client hosts can submit jobs to an LSF cluster, but cannot execute jobs sent from other hosts. If this field is not defined, then the default is 1.

RESOURCES - boolean resources available on this host. The resource names are strings defined in the Resource section of the file `lsf.shared`. You may list any number of resources, enclosed in parentheses and separated by blanks or tabs. For example, `(fs frame hpux)`.

The `lsf.sudoers` File

This file allows a list of permitted users to perform certain privileged operations in the LSF cluster as either the superuser or any other designated user. This file is optional.

The `lsf.sudoers` file must be located in `/etc` and it must be owned by root.

The format of this file is very similar to that of the `lsf.conf` file (see *'The lsf.conf File'* on page 75). Each line of the file is a `NAME=VALUE` statement, where `NAME` describes an authorized operation and `VALUE` is a single string or multiple strings enclosed in quotes. Lines starting with '#' are comments and are ignored.

The currently recognized variables in this file include:

LSF_STARTUP_USERS

This parameter is used to enable a list of specified users to start up LSF daemons as `root` using the LSF administrative commands `lsadmin` and `badmin`.

UNIX By default, the superuser is the only user who can start up the LSF as `root`.

Note that `lsadmin` and `badmin` must be installed as *setuid root* programs for this to work. Possible values for this variable include:

`all_admins` - this allows all LSF administrators configured in the `lsf.cluster.`*`cluster`* file to start up LSF daemons as `root` by running the `lsadmin` and `badmin` commands.

`user1 user2` - this allows listed user(s) to perform the startup operations. If this list contains more than one user, it must be enclosed with quotes. For example:

```
LSF_STARTUP_USERS="user1 user2"
```

CAUTION!

Defining LSF_STARTUP_USERS as `all_admins` incurs some security risk because administrators can be configured by a primary LSF administrator who is not root. You should explicitly list the login names of all authorized administrators here so that you have full control of who can start daemons as root.

LSF_STARTUP_PATH

The absolute pathname of the directory where the server binaries, namely, `lim`, `res`, `sbatchd`, are installed. This is normally LSF_SERVERDIR as defined in your `lsf.conf` file. LSF will allow the users defined in LSF_STARTUP_USERS to start the daemons installed in the LSF_STARTUP_PATH directory as `root`.

Note

Both LSF_STARTUP_USERS *and* LSF_STARTUP_PATH *must be defined for this feature to work.*

LSB_PRE_POST_EXEC_USER

This parameter defines the authorized user for the LSF JobScheduler queue level pre-execution and post-execution commands. These commands can be configured at the queue level by the LSF administrator. If `LSB_PRE_POST_EXEC_USER` is defined, the queue level pre-execution and post-execution commands will be run as the user defined. If this parameter is not defined, the commands will be run as the user who submitted the job. In particular, you can define this variable if you need to run commands as `root`.

See *'Pre- and Post-Execution Commands'* on page 17 for details of pre-execution and post-execution.

You can only define a single user name in this parameter.

LSF_EAUTH_USER

This defines the user name to run the external authentication executable, `eauth`. If this is parameter is not defined, then `eauth` will be run as the primary LSF administrator.

LSF_EEXEC_USER

This defines the user name to run the external execution command, `eexec`. If this parameter is not defined, then `eexec` will be run as the user who submitted the job.

5. LSF JobScheduler Configuration Reference

This chapter describes the LSF JobScheduler configuration files `lsb.params`, `lsb.hosts`, `lsb.queues`, `lsb.alarms`, and `lsb.calendars`. These files use the same horizontal and vertical section structure as the LIM configuration files (see '*Configuration File Formats*' on page 29). All LSF JobScheduler configuration files are found in the `LSB_CONFDIR/`*cluster*`/configdir` directory.

The `lsb.params` File

The `lsb.params` file defines general parameters used by the LSF JobScheduler cluster. This file contains only one section.

Most of the parameters that can be defined in the `lsb.params` file control timing within the LSF JobScheduler system. The default settings provide good throughput for long-running batch jobs while adding a minimum of processing overhead to the daemons.

Parameters

This section and all the keywords in this section are optional. If keywords are not present, LSF JobScheduler assumes default values for the corresponding keywords.

The valid keywords for this section are:

DEFAULT_QUEUE = ***queue***

DEFAULT_QUEUE lists a name for the LSF JobScheduler queue defined in the lsb.queues file. When a user submits a job to the LSF JobScheduler system without explicitly specifying a queue and the user's environment variable LSB_DEFAULTQUEUE is not set, LSF JobScheduler queues the job in the default queue.

If this keyword is not present or no valid value is given, then LSF JobScheduler automatically creates a default queue named default with all the default parameters (see *'The lsb.queues File'* on page 97).

MBD_SLEEP_TIME = ***integer***

The LSF JobScheduler job dispatching interval. It determines how often the LSF JobScheduler system tries to dispatch pending jobs.

Default: 60 (seconds)

SBD_SLEEP_TIME = ***integer***

The LSF JobScheduler job checking interval. It determines how often the LSF JobScheduler system checks the load conditions of each host to decide whether jobs on the host must be suspended or resumed.

Default: 30 (seconds)

JOB_ACCEPT_INTERVAL = ***integer***

The number of MBD_SLEEP_TIME periods to wait after dispatching a job to a host, before dispatching a second job to the same host. If JOB_ACCEPT_INTERVAL is zero, a host may accept more than one job in each job dispatching interval (MBD_SLEEP_TIME).

Default: 1

MAX_SBD_FAIL = ***integer***

The maximum number of retries for reaching a non-responding slave batch daemon, sbatchd. The interval between retries is defined by MBD_SLEEP_TIME. If the master batch daemon fails to reach a host, and has retried `MAX_SBD_FAIL` times, the host is considered unavailable.

Default: 3

`CLEAN_PERIOD =` ***integer***

The amount of time that job records for jobs that have finished or have been killed remain in 'DONE' or 'EXIT' status before they are either cleaned out of `mbatchd`'s memory or moved into 'PEND' status for next schedule. Users can still see all finished jobs after they have finished using the `bjobs` command. For jobs that finished more than `CLEAN_PERIOD` seconds ago (and as a result are cleaned out of memory), use the `bhist` command.

Default: 3600 (seconds)

`MAX_JOB_NUM =` ***integer***

The maximum number of finished jobs whose events are to be stored in an event log file (see the `lsb.events(5)` manual page). Once the limit is reached, the `mbatchd` switches the event log file. See *'LSF JobScheduler Event Log'* on page 52.

Default: 1000

The `lsb.hosts` File

The `lsb.hosts` file contains host related configuration information for the batch server hosts in the cluster. This file is optional.

Host Section

The optional `Host` section contains per-host configuration information. Each host, host model or host type can be configured to run a maximum number of jobs. Hosts, host models or host types can also be configured to run jobs only under specific load conditions.

If no hosts, host models or host types are named in this section, LSF JobScheduler uses all hosts in the LSF cluster as server hosts. Otherwise, only the named hosts, host models and host types are used by LSF JobScheduler. If a line in the `Host` section lists the reserved host name `default`, LSF JobScheduler uses all hosts in the cluster and the settings on that line apply to every host not referenced in the section, either explicitly or by listing its model or type.

The first line of this section gives the keywords that apply to the rest of the lines. The keyword `HOST_NAME` must appear. Other supported keywords are optional.

HOST_NAME

The name of a host defined in the `lsf.cluster.`*`cluster`* file, a host model or host type defined in the `lsf.shared` file, or the reserved word `default`.

MXJ

The maximum number of job slots for the host. On multiprocessor hosts `MXJ` should be set to at least the number of processors to fully use the CPU resource.

Default: unlimited.

r15s, r1m, r15m, ut, pg, io, ls, it, tmp, swp, mem, name

Scheduling and suspending thresholds for the dynamic load indices supported by LIM. Each load index column must contain either the default entry or two numbers separated by a slash '/', with no white space. The first number is the scheduling threshold for the load index; the second number is the suspending threshold. See *Section 4, 'Resources', beginning on page 45* of the *LSF JobScheduler User's Guide* for complete descriptions of the load indices.

Default: no threshold.

Host Groups

The `HostGroup` section is optional. This section defines names for sets of hosts. The host group name can then be used in other host group, and queue definitions, as well as on an LSF JobScheduler command line. When a host group name is used, it has exactly the same effect as listing all of the host names in the group.

The host group section must begin with a line containing the mandatory keywords `GROUP_NAME` and `GROUP_MEMBER`. Each other line in this section must contain an alphanumeric string for the group name, and a list of host names or previously defined group names enclosed in parentheses and separated by white space.

Host names and host group names can appear in more than one host group. The reserved name `all` specifies all hosts in the cluster.

```
Begin HostGroup
GROUP_NAME  GROUP_MEMBER
licence1   (hostA hostD)
sys_hosts  (hostF license1 hostK)
End HostGroup
```

This example section defines two host groups. The group license1 contains the hosts *hostA* and *hostD*; the group sys_hosts contains *hostF* and *hostK*, along with all hosts in the group license1. Group names must not conflict with host names.

The `lsb.queues` File

The `lsb.queues` file contains definitions of the queues in an LSF cluster. This file is optional. If no queues are configured, LSF JobScheduler creates a queue named default, with all parameters set to default values (see the description of `DEFAULT_QUEUE` in *'The lsb.params File'* on page 93).

Queue definitions are horizontal sections that begin with the line `Begin Queue` and end with the line `End Queue`. You can define at most 40 queues in an LSF cluster. Each queue definition contains the following parameters:

General Parameters

QUEUE_NAME = string

The name of the queue. This parameter must be defined, and has no default. The queue name can be any string of non-blank characters up to 40 characters long. It is best to use

6 to 8 character long names made up of letters, digits, and possibly underscores '_' or dashes '-'.

PRIORITY = integer

This parameter indicates the priority of the queue relative to other LSF Batch queues. Note that this is an LSF JobScheduler dispatching priority, completely independent of the operating system's priority system for time-sharing processes.

LSF JobScheduler tries to schedule jobs from queues with larger `PRIORITY` values first. This does not mean that jobs in lower priority queues are not scheduled unless higher priority queues are empty. Higher priority queues are checked first, but not all jobs in them are necessarily scheduled. For example, a job might be held because no machine with the right resources is available. Lower priority queues are then checked and, if possible, their jobs are scheduled.

If more than one queue is configured with the same `PRIORITY`, LSF JobScheduler schedules jobs from all these queues in first-come, first-served order.

Default: 1

HOSTS = name(+pref_level) ...

The list of hosts on which jobs from this queue can be run. Each name in the list must be a valid host name, host group name or host partition name as configured in the `lsb.hosts` file. The name can be optionally followed by +*pref_level* to indicate the preference for dispatching a job to that host, host group, or host partition. *pref_level* is a positive number specifying the preference level of that host. If a host preference is not given, it is assumed to be 0.

Hosts at the same level of preference are ordered by load. For example:

```
HOSTS = hostA+1 hostB hostC+1 servers+3
```

where *servers* is a host group name referring to all computer servers. This defines three levels of preferences: run jobs on *servers* as much as possible, or else on *hostA* and *hostC*. Jobs should not run on *hostB* unless all other hosts are too busy to accept more jobs.

If you use the reserved word ’others’, it means jobs should run on all hosts not explicitly listed. You do not need to define this parameter if you want to use all JobScheduler server hosts and you do not need host preferences.

Queue-Level Pre-/Post-Execution Commands

Pre- and post-execution commands can be configured on a per-queue basis. These commands are run on the execution host before and after a job from this queue is run, respectively. By configuring appropriate pre- and/or post-execution commands various situations can be handled such as:

- Creating and deleting scratch directories for the job
- Assigning jobs to run on specific processors on SMP machines
- Customized scheduling
- License availability checking

Note that the job-level pre-exec specified with the `-E` option of `bsub` is also supported. In some situations (for example, license checking) it is possible to specify a queue-level pre-execution command instead of requiring every job be submitted with the `-E` option.

The execution commands are specified using the `PRE_EXEC` and `POST_EXEC` keywords; for example:

```
Begin Queue
QUEUE_NAME     = priority
PRIORITY       = 43
PRE_EXEC       = /usr/people/lsf/pri_prexec
POST_EXEC      = /usr/people/lsf/pri_postexec
End Queue
```

The following points should be considered when setting up the pre- and post-execution commands for queues:

- The entire contents of the configuration line of the pre- and post-execution commands are run under `/bin/sh -c`, so shell features can be used in the command. For example, the following is valid:

```
PRE_EXEC = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC = /usr/local/lsf/misc/testq_post | grep -v "Hey!"
```

- Both the pre- and post-execution commands are run as the user by default. If you must run these commands as a different user, such as root (to do privileged operations, if necessary), you can configure the parameter `LSB_PRE_POST_EXEC_USER` in the `lsf.sudoers` file. See *'The lsf.sudoers File'* on page 89 for details.

- The pre- and post-execution commands are run in `/tmp`.

- Standard input and standard output and error are set to `/dev/null`. The output from the pre- and post-execution commands can be explicitly redirected to a file for debugging purposes.

- If the pre-execution command exits with a non-zero exit code, then it is considered to have failed and the job is requeued to the head of the queue. This feature can be used to implement customized scheduling by having the pre-execution command fail if conditions for dispatching the job are not met.

- The `PATH` environment variable is set to '`/bin /usr/bin /sbin /usr/sbin`'.

- Other environment variables set for the job are also set for the pre- and post-execution commands.

- When a job is dispatched from a queue which has a pre-execution command, LSF Batch will remember the post-execution command defined for the queue from which the job is dispatched. If the job is later switched to another queue or the post-execution command of the queue is changed, LSF Batch will still run the original post-execution command for this job.

- When the post-execution command is run, the environment variable, `LSB_JOBEXIT_STAT`, is set to the exit status of the job. Refer to the manual page for `wait(2)` for the format of this exit status.

- The post-execution command is also run if a job is requeued because the job's execution environment fails to be set up. The environment variable, `LSB_JOBPEND`, is set if the job is requeued. If the job's execution environment could not be set up, `LSB_JOBEXIT_STAT` is set to `0`.
- If both queue and job level pre-execution commands are specified, the job level pre-execution is run after the queue level pre-execution command.

Default: no pre- and post-execution commands

Job Starter

A Job starter can be defined for each queue to bring the actual job into the desired environment before execution. The configuration syntax for job starter is:

```
JOB_STARTER = starter
```

Here, the starter string is any executable that can be used to start the job command line.

UNIX When LSF JobScheduler runs the job, it executes `/bin/sh -c "JOB_STARTER job_command_line"`. Thus a job starter can be anything that can be run together with the job command line.

The `lsb.alarms` File

This file defines all alarms in LSF JobScheduler and how alarms should be handled. An alarm provides a mechanism for sending a notification of an alert condition. An alarm is associated with a job when the job is defined. Users can specify when an alarm should be triggered for the job. The alarm definition in this file is read by the `raisealarm` command that is invoked by the LSF JobScheduler when certain job exceptions are detected. Users can view the alarm definition through the `xbalarms` GUI or `balarms` command. For alarms which require periodic notifications, `alarmd` will read this file to invoke the appropriate notification method.

The `lsb.alarms` file consists of multiple "Alarm" sections where each section corresponds to one alarm. Each section has a number of keywords which are used to define the alarm parameters. Each alarm parameter can be specified in the “Alarm” section:

NAME

The name of the alarm. Use any ASCII string with up to 32 characters. The word ‘default’ is reserved and cannot be used as an alarm name.

NOTIFICATION

The notification method to be used to inform users or administrators that the alarm has been triggered. The notification method can be either email or by invoking a site-defined executable. To send the notifcation through email, the value should be:

```
EMAIL[userlist]
```

Here, `userlist` is a list of one or more login names separated by spaces. To invoke a site-defined executable, the value should be of the form:

```
CMD[cmdname arguments]
```

In this case, `cmdname` is the name of the command to be invoked and `arguments` are any arguments supplied to the command (optionally).

NOTIFICATION_RETRY

This parameter controls how often the notification is sent if an alarm incident is not acknowledged. A limit can be set on the maximum number of notifications. The format of this parameter is:

`retry_interval` or `retry_interval/max_retry`

Here, `retry_interval` is the number of minutes between renotifications and `max_retry` is the maximum number of renotifications. If `max_retry` is not specified then renotifications will be sent until the alarm is acknowledged or until the alarm expires.

By default, if NOTIFICATION_RETRY is not specified, the notification method will only be invoked once.

EXPIRATION

This parameter specifies the expiration time for an alarm incident in minutes. If an incident is still in the open state after the specified time, its state will automatically be changed to expired and be moved to the alarm history file. This is intended to prevent low-priority alarms from filling up the alarm log indefinitely.

By default, the expiration time is infinite.

DESCRIPTION

A brief description of the alarm.

Examples

The following are examples of an alarm definition:

```
Begin Alarm
NAME=DBError
DESCRIPTION=Send Administrator a Page on errors in DBMS
NOTIFICATION=CMD[/usr/local/bin/sendpage dbadmin]
NOTIFICATION_RETRY=30/5
End Alarm

Begin Alarm
NAME=DiskFull
DESCRIPTION=Send LSF administrator a mail when full disk is detected
NOTIFICATION=EMAIL[lsfadmin]
EXPIRATION=40
End Alarm
```

The `lsb.calendars` File

This file contains the definitions of system calendars. System calendars are read-only calendars which can be referenced by all users. The use of system calendars is discussed in '*System Calendars*' on page 66.

The `lsb.calendars` file consists of multiple “Calendar” sections, each corresponding to a single calendar. Each calendar section requires the NAME and CAL_EXPR parameter and can optionally contain a DESCRIPTION parameter.

A “Calendar” section in the `lsb.calendars` file is structured as follows:

```
Begin Calendar
NAME=name
CAL_EXPR=calendar expression
DESCRIPTION=description
End Calendar
```

The NAME parameter is a character string that names the system calendar.

The syntax of the CAL_EXPR parameter is described in the *LSF JobScheduler User’s Guide* and the `bcadd(1)` man page.

A. Troubleshooting and Error Messages

This chapter describes some common problems with LSF and LSF JobScheduler operations, answers some frequently asked questions, and provides some instructions for solving problems.

Error Log Messages

When something goes wrong, the daemons almost always log an error message. The first step is to find the appropriate log and see whether there are any messages.

Specific error log messages are listed in '*Error Messages*' on page 111.

Finding the Error Logs

Error messages of LSF servers are logged in either the `syslog(3)` or to files. This is determined by the `LSF_LOGDIR` definition in the `lsf.conf` file. For complete instructions on finding the LSF server logs, see '*Managing Error Logs*' on page 45.

UNIX

If you configure LSF to log daemon messages using syslog, the destination file is determined by the syslog configuration. On most systems, you can find out which file the LSF messages are logged in with the command:

```
grep daemon /etc/syslog.conf
```

Once you have found the syslog file, you can select the LSF error messages with the command:

```
egrep 'lim|res|batchd' syslog_file
```

Look at the `/etc/syslog.conf` file and the manual page for `syslog` or `syslogd` for help in finding the system logs.

When searching for log messages from LSF servers, you are more likely to find them on the remote machine where LSF put the task than on your local machine where the command was given.

LIM problems are usually logged on the master host. Run `lsid` to find the master host, and check `syslog` or the `lim.log.`*hostname* file on the master host. The `res.log.`*hostname* file contains messages about RES problems, execution problems and setup problems for LSF. Most problems with interactive applications are logged in the remote machine's log files.

Errors from LSF JobScheduler are logged either in the `mbatchd.log.`*hostname* file on the master host, or the `sbatchd.log.`*hostname* file on the execution host. The `bjobs` or `bhist` command tells you the execution host for a specific job.

Most LSF log messages include the name of an internal LSF function to help the developers locate problems. Many error messages can be generated in more than one place, so it is important to report the entire error message when you ask for technical support.

Shared File Access

UNIX

Shared File Access on UNIX

A frequent problem with LSF is non-accessible files due to a non-uniform file space. If a task is run on a remote host where a file it requires cannot be accessed using the same name, an error results. Almost all interactive LSF commands fail if the user's current working directory cannot be found on the remote host.

If you are running NFS, rearranging the NFS mount table may solve the problem. If your system is running the `automount` server, LSF tries to map the file names, and in most cases it succeeds. If shared mounts are used, the mapping may break for those files. In such cases, specific measures need to be taken to get around it.

The automount maps must be managed through NIS. When LSF tries to map file names, it assumes that automounted file systems are mounted under the `/tmp_mnt` directory.

NT

Shared File Access on Windows NT

To share files among NT machines, set up the server and access it from the client. You can access files on the share either by specifying a UNC path (\\server\share\path) or connecting the share to a local drive name and using `drive:path` syntax. Using UNC is recommended because drive mappings may be different across machines while UNC allows you to unambiguously refer to a file on the network.

Shared Files Across UNIX and NT

For file sharing across UNIX and NT, you require a third-party NFS product on NT to export directories from NT to UNIX.

Common LSF Base Problems

This section lists some other common problems with the LIM, the RES and interactive applications.

LIM Dies Quietly

Run the following command to check for errors in the LIM configuration files.

```
% lsadmin ckconfig -v
```

This displays most configuration errors. If this does not report any errors, check in the LIM error log.

LIM Unavailable

Sometimes the LIM is up but `lsload` prints the following error message.

```
Communication time out.
```

If the LIM has just been started, this is normal, because the LIM needs time to get initialized by reading configuration files and contacting other LIMs.

If the LIM does not become available within one or two minutes, check the LIM error log on the local host.

When the local LIM is running but there is no master LIM in the cluster, LSF applications display the following message.

```
Cannot locate master LIM now, try later.
```

Check the LIM error logs on the first few hosts listed in the “Host” section of the `lsf.cluster.`*cluster* file.

RES Does Not Start

Check the RES error log.

NT If the RES is unable to read the `lsf.conf` file and does not know where to write error messages, it logs errors into `C:\temp`.

UNIX If the RES is unable to read the `lsf.conf` file and does not know where to write error messages, it logs errors into `syslog(3)`.

User Permission Denied

If remote execution fails with the following error message, the remote host could not securely determine the user ID of the user requesting remote execution.

```
User permission denied.
```

Check the RES error log on the remote host; this usually contains a more detailed error message.

If you are not using an identification daemon (`LSF_AUTH` is not defined in the `lsf.conf` file), then all applications that do remote executions must be owned by *root* with the `setuid` bit set. This can be done as follows.

```
% chmod 4755 filename
```

If the binaries are on an NFS-mounted file system, make sure that the file system is not mounted with the `nosuid` flag.

If you are using an identification daemon (defined in the `lsf.conf` file by the variable LSF_AUTH), `inetd` must be configured to run the daemon. The identification daemon must not be run directly.

If LSF_USE_HOSTEQUIV is defined in the `lsf.conf` file, check if `/etc/hosts.equiv` or `HOME/.rhosts` on the destination host has the client host name in it. Inconsistent host names in a name server with `/etc/hosts` and `/etc/hosts.equiv` can also cause this problem.

On SGI hosts running a name server, you can try:

```
% setenv HOSTRESORDER = local,nis,bind
```

This tells the host name lookup code to search the `/etc/hosts` file before calling the name server.

Non-uniform File Name Space

A command may fail with the following error message.

```
chdir(...) failed: no such file or directory
```

This is due to a non-uniform file name space. You are trying to execute a command remotely, where either your current working directory does not exist on the remote host, or your current working directory is mapped to a different name on the remote host.

If your current working directory does not exist on a remote host, you should not execute commands remotely on that host.

UNIX If the directory exists, but is mapped to a different name on the remote host, you have to create symbolic links to make them consistent.

LSF can resolve most, but not all, problems using `automount`. The automount maps must be managed through NIS. Follow the instructions in your Release Notes for obtaining technical support if you are running automount and LSF is not able to locate directories on remote hosts.

Common LSF JobScheduler Problems

This section lists some common problems with LSF JobScheduler. Most problems are due to incorrect installation or configuration. Check the `mbatchd` and `sbatchd` error log files; often the log messages points directly to the problem.

Batch Daemons Die Quietly

First, check the `sbatchd` and `mbatchd` error logs. Try running the following command to check the configuration.

```
% badmin ckconfig
```

This reports most errors. You should also check if there is any email from LSF JobScheduler in the LSF administrator’s mail box. If the `mbatchd` is running but the `sbatchd` dies on some hosts, it may be because `mbatchd` has not been configured to use those hosts. See ‘*Host Not Used By LSF JobScheduler*’ on page 111.

`sbatchd` Starts But `mbatchd` Does Not

Check whether LIM is running. You can test this by running the `lsid` command. If LIM is not running properly, follow the suggestions in this chapter to fix the LIM first. You should make sure that LSF and LSF JobScheduler are using the same `lsf.conf` file. It is possible that `mbatchd` is temporarily unavailable because the master LIM is temporarily unknown.

```
sbatchd: unknown service
```

Check whether services are registered properly. See ‘*Registering LSF Service Ports*’ on page 84 of the *LSF Installation Guide*.

Host Not Used By LSF JobScheduler

If you configure a list of server hosts in the “Host” section of the `lsb.hosts` file, `mbatchd` allows `sbatchd` to run only on the hosts listed. If you try to configure an unknown host in the “HostGroup” or “HostPartition” sections of the `lsb.hosts` file, or as a `HOSTS` definition for a queue in the `lsb.queues` file, `mbatchd` logs the message:

```
mbatchd on hostA: LSB_CONFDIR/cluster/configdir/file(line #):
Host hostB is not used by lsbatch;

ignored
```

If you start `sbatchd` on a host that is not known by `mbatchd`, `mbatchd` rejects the `sbatchd`. The `sbatchd` logs the following message and exits.

```
This host is not used by lsbatch system.
```

Both of these errors are most often caused by forgetting to run the following commands, in order, after adding a host to the configuration.

```
% lsadmin reconfig

% badmin reconfig
```

You must run both of these before starting the daemons on the new host.

Error Messages

The following error messages are logged by the LSF daemons, or displayed by the `lsadmin ckconfig` and `badmin ckconfig` commands. LSF daemon message logs are described in ‘*Managing Error Logs*’ on page 45.

General Errors

The messages listed in this section may be generated by any LSF daemon.

`can’t open file: error`

The daemon could not open the named file for the reason given by `error`. This error is usually caused by incorrect file permissions or missing files. All directories in the path to the configuration files must have ‘x’ permission for the LSF administrator, and the actual files must have ‘r’ permission. Missing files could be caused by incorrect path names in the `lsf.conf` file, running LSF daemons on a host where the configuration files have not been installed, or having a symbolic link pointing to a nonexistent file or directory.

`file(line): malloc failed`

Memory allocation failed. Either the host does not have enough available memory or swap space, or there is an internal error in the daemon. Check the program load and available swap space on the host; if the swap space is full, you must add more swap space or run fewer (or smaller) programs on that host.

`auth_user: getservbyname(ident/tcp) failed: error;`
`ident must be registered in services`

`LSF_AUTH=ident` is defined in the `lsf.conf` file, but the `ident/tcp` service is not defined in the services database. Add `ident/tcp` to the services database, or remove LSF_AUTH from the `lsf.conf` file and `setuid root` those LSF binaries that require authentication.

`auth_user: operation(<host>/<port>) failed: error`

`LSF_AUTH=ident` is defined in the `lsf.conf` file, but the LSF daemon failed to contact the ident daemon on `host`. Check that `ident` is defined in `host`’s `inetd.conf` and the ident daemon is running on `host`.

`auth_user: Authentication data format error (rbuf=<data>) from <host>/<port>`

`auth_user: Authentication port mismatch (...) from <host>/<port>`

`LSF_AUTH=ident` is defined in `lsf.conf`, but there is a protocol error between LSF and the ident daemon on `host`. Make sure the ident daemon on `host` is configured correctly.

`userok: Request from bad port (<portno>), denied`

LSF_AUTH is not defined, and the LSF daemon received a request that originates from a non-privileged port. The request is not serviced.

Set the LSF binaries (e.g., `lsrun`) to be owned by root with the `setuid` bit set, or define `LSF_AUTH=ident` and set up an ident server on all hosts in the cluster. If the binaries are on an NFS-mounted file system, make sure that the file system is not mounted with the `nosuid` flag.

`userok: Forged username suspected from <host>/<port>: <claimed user>/<actual user>`

The service request claimed to come from user `claimed user` but ident authentication returned that the user was actually `actual user`. The request was not serviced.

`userok: ruserok(<host>,<uid>) failed`

`LSF_USE_HOSTEQUIV` is defined in the `lsf.conf` file but `host` has not been set up as an equivalent host (see `/etc/host.equiv`), and user `uid` has not set up a `.rhosts` file.

`init_AcceptSock: RES service(res) not registered, exiting`

`init_AcceptSock: res/tcp: unknown service, exiting`

`initSock: LIM service not registered. See LSF Guide for help`

`initSock: Service lim/udp is unknown. Read LSF Guide for help`

`get_ports: <serv> service not registered`

The LSF services are not registered. See *'Registering LSF Service Ports'* on page 84 of the *LSF Installation Guide*.

`init_AcceptSock: Can't bind daemon socket to port <port>: error, exiting`

`init_ServSock: Could not bind socket to port <port>: error`

These error messages can occur if you try to start a second LSF daemon (e.g., RES is already running, and you execute RES again). If this is the case, and you want to start the new daemon, kill the running daemon or use the `lsadmin` or `badmin` commands to shut down or restart the daemon.

Configuration Errors

The messages listed in this section are caused by problems in the LSF configuration files. General errors are listed first, and then errors from specific files.

`file(line): Section name expected after Begin; ignoring section`

`file(line): Invalid section name name; ignoring section`

The keyword `begin` at the specified line is not followed by a section name, or is followed by an unrecognised section name.

`file(line): section section: Premature EOF`

The end of file was reached before reading the `end section` line for the named section.

`file(line): keyword line format error for section section; Ignore this section`

The first line of the section should contain a list of keywords. This error is printed when the keyword line is incorrect or contains an unrecognised keyword.

`file(line): values do not match keys for section section; Ignoring line`

The number of fields on a line in a configuration section does not match the number of keywords. This may be caused by forgetting to put `()` in a column to represent the default value.

`file: HostModel section missing or invalid`

`file: Resource section missing or invalid`

`file: HostType section missing or invalid`

The `HostModel`, `Resource`, or `HostType` section in the `lsf.shared` file is either missing or contains an unrecoverable error.

`file(line): Name name reserved or previously defined. Ignoring index`

The name assigned to an external load index must not be the same as any built-in or previously defined resource or load index.

`file(line): Duplicate clustername name in section cluster Ignoring current line.`

A cluster name is defined twice in the same `lsf.shared` file. The second definition is ignored.

***fil*`e(line): Bad cpuFactor for host model model. Ignoring line`**

The CPU factor declared for the named host model in the `lsf.shared` file is not a valid number.

`file(line): Too many host models, ignoring model name`

You can declare at most 25 host models in the `lsf.shared` file.

`file(line): Resource name name too long in section resource. Should be less than 40 characters. Ignoring line.`

The maximum length of a resource name is 39 characters. Choose a shorter name for the resource.

`file(line): Resource name name reserved or previously defined. Ignoring line.`

You have attempted to define a resource name that is reserved by LSF or already defined in the `lsf.shared` file. Choose another name for the resource.

`file(line): illegal character in resource name: name, section resource. Line ignored.`

Resource names must begin with a letter in the set [a-zA-Z], followed by letters, digits or underscores [a-zA-Z0-9_].

LIM Messages

The following messages are logged by the LIM.

`main: LIM cannot run without licenses, exiting`

The LSF software license key is not found or has expired. Check that FLEXlm is set up correctly, or contact your LSF technical support.

`main: Received request from unlicensed host <host>/<port>`

LIM refuses to service requests from hosts that do not have licenses. Either your LSF license has expired, or you have configured LSF on more hosts than your license key allows.

`initLicense: Trying to get license for LIM from source <LSF_CONFDIR/license.dat>`

`getLicense: Can’t get software license for LIM from license file <LSF_CONFDIR/license.dat>: feature not yet available.`

Your LSF license is not yet valid. Check whether the system clock is correct.

`findHostbyAddr/<proc>: Host <host>/<port> is unknown by <myhostname>`

`function: Gethostbyaddr_(<host>/<port>) failed: error`

`main: Request from unknown host <host>/<port>: error`

`function: Received request from non-LSF host <host>/<port>`

The daemon does not recognize `host` as an LSF host. The request is not serviced. These messages can occur if `host` was added to the configuration files, but not all the daemons have been reconfigured to read the new information. If the problem still occurs after reconfiguring all the daemons, check whether `host` is a multi-addressed host. See the LSF Installation Guide for more information on host naming.

`rcvLoadVector: Sender (<host>/<port>) may have different config?`

`MasterRegister: Sender (host) may have different config?`

LIM detected inconsistent configuration information with the sending LIM. Run the following command so that all the LIMs have the same configuration information.

```
lsadmin reconfig
```

Note any hosts that failed to be contacted.

`rcvLoadVector: Got load from client-only host <host>/<port>. Kill LIM on <host>/<port>`

A LIM is running on an LSF client host. Run `lsadmin limshutdown` *host*, or go to the client host and kill the LIM daemon.

`saveIndx: Unknown index name <name> from ELIM`

LIM received an external load index name that is not defined in the `lsf.shared` file. If `name` is defined in the `lsf.shared` file, reconfigure the LIM. Otherwise, add *name* to `lsf.shared` and reconfigure all the LIMs.

`saveIndx: ELIM over-riding value of index <name>`

This is a warning message. The ELIM sent a value for one of the built-in index names. LIM uses the value from ELIM in place of the value obtained from the kernel.

`getusr: Protocol error numIndx not read (cc=num): error`

`getusr: Protocol error on index number (cc=num): error`

Protocol error between ELIM and LIM. See *'Changing LIM Configuration'* on page 55 for a description of the protocol.

RES Messages

These messages are logged by the RES.

`doacceptconn: getpwnam(<username>@<host>/<port>) failed: error`

`doacceptconn: User <username> has uid <uid1> on client host <host>/<port>, uid <uid2> on RES host; assume bad user`

`authRequest: username/uid <userName>/<uid>@<host>/<port> does not exist`

`authRequest: Submitter's name <clname>@<clhost> is different from name <lname> on this host`

RES assumes that a user has the same user ID and user name on all the LSF hosts. These messages occur if this assumption is violated. If the user is allowed to use LSF for interactive remote execution, make sure the user's account has the same user ID and user name on all LSF hosts.

`doacceptconn: root remote execution permission denied`

`authRequest: root job submission rejected`

Root tried to execute or submit a job but LSF_ROOT_REX is not defined in the `lsf.conf` file.

`resControl: operation permission denied, uid = <uid>`

The user with user ID `uid` is not allowed to make RES control requests. Only the LSF manager, or `root` if LSF_ROOT_REX is defined in the `lsf.conf` file, can make RES control requests.

`resControl: access(respath, X_OK): error`

The RES received a reboot request, but failed to find the file `respath` to reexecute itself. Make sure `respath` contains the RES binary, and it has execution permission.

LSF JobScheduler Messages

The following messages are logged by the `mbatchd` and `sbatchd` daemons.

`renewJob: Job <jobId>: rename(<from>,<to>) failed: error`

`mbatchd` failed in trying to re-submit a rerunnable job. Check that the file `from` exists and that the LSF administrator can rename the file. If `from` is in an AFS directory, check that the LSF administrator's token processing is properly set up (see *'Installation on AFS'* on page 97 of the *LSF Installation Guide*).

`logJobInfo_: fopen(<logdir/info/jobfile>) failed: error`

`logJobInfo_: write <logdir/info/jobfile> <data> failed: error`

`logJobInfo_: seek <logdir/info/jobfile> failed: error`

`logJobInfo_: write <logdir/info/jobfile> xdrpos <pos> failed: error`

`logJobInfo_: write <logdir/info/jobfile> xdr buf len <len> failed: error`

`logJobInfo_: close(<logdir/info/jobfile>) failed: error`

`rmLogJobInfo: Job <jobId>: can't unlink(<logdir/info/jobfile>): error`

`rmLogJobInfo_: Job <jobId>: can't stat(<logdir/info/jobfile>): error`

`readLogJobInfo: Job <jobId> can't open(<logdir/info/jobfile>): error`

`start_job: Job <jobId>: readLogJobInfo failed: error`

`readLogJobInfo: Job <jobId>: can't read(<logdir/info/jobfile>) size size: error`

`initLog: mkdir(<logdir/info>) failed: error`

`<fname>: fopen(<logdir/file> failed: error`

`getElogLock: Can't open existing lock file <logdir/file>: error`

`getElogLock: Error in opening lock file <logdir/file>: error`

`releaseElogLock: unlink(<logdir/lockfile>) failed: error`

`touchElogLock: Failed to open lock file <logdir/file>: error`

`touchElogLock: close <logdir/file> failed: error`

`mbatchd` failed to create, remove, read, or write the log directory or a file in the log directory, for the reason given in `error`. Check that LSF managerid has read, write, and execute permissions on the `logdir` directory.

If `logdir` is on AFS, check that the instructions in *'Installation on AFS'* on page 97 of the *LSF Installation Guide* have been followed. Use the following command:

```
% fs la
```

to verify that the LSF administrator owns `logdir` and that the directory has the correct `acl`.

`replay_newjob: File <logfile> at line <line>: Queue <queue> not found, saving to queue <lost_and_found>`

`replay_switchjob: File <logfile> at line <line>: Destination queue <queue> not found, switching to queue <lost_and_found>`

When `mbatchd` was reconfigured, jobs were found in `queue` but that queue is no longer in the configuration.

`replay_startjob: JobId <jobId>: exec host <host> not found, saving to host <lost_and_found>`

When `mbatchd` was reconfigured, the event log contained jobs dispatched to `host`, but that host is no longer configured to be used by LSF JobScheduler.

`do_restartReq: Failed to get hData of host <hostname>/<hostaddr>`

`mbatchd` received a request from `sbatchd` on host *hostname*, but that host is not known to `mbatchd`. Either the configuration file has been changed but `mbatchd` has not been reconfigured to pick up the new configuration, or *hostname* is a client host but the `sbatchd` daemon is running on that host. Run the `badmin reconfig` command to reconfigure the `mbatchd` or kill the `sbatchd` daemon on *hostname*.

B. LSF Directories

This table lists the directories used by the LSF system, their modes and contents.

Table 1. LSF Directories

Directory	Mode	Contents
`$LSB_CONFDIR` `$LSB_CONFDIR/*`	755	LSF JobScheduler configuration files, must be owned by the primary LSF administrator, and shared by all potential master hosts
`$LSB_SHAREDIR/`*`cluster`*`/logdir`	755	LSF JobScheduler accounting files, must be owned by the primary LSF administrator, and shared by all potential master hosts
`$LSF_BINDIR`	755	User commands, must allow setuid to root, shared by all hosts of the same type
`$LSF_CONFDIR`	755	LSF cluster configuration files, must be owned by the primary LSF administrator, and shared by all LSF server hosts
`$LSF_ENVDIR`	755	`lsf.conf` file, must be owned by root
`$LSF_INCLUDEDIR`	755	Header files `lsf/lsf.h`
`$LSF_INDEP`	755	Host type independent files shared by all hosts
`$LSF_LIBDIR`	755	LSF libraries, shared by all hosts of the same type
`$LSF_LOGDIR`	1777	Server error logs, must be owned by root
`$LSF_MACHDEP`	755	Host type dependent files shared by all hosts of the same type
`$LSF_MANDIR`	755	LSF `man` pages shared by all hosts
`$LSF_MISC`	755	Examples and other miscellaneous files shared by all hosts

B LSF Directories

Table 1. LSF Directories

Directory	Mode	Contents
`$LSF_SERVERDIR`	755	Server binaries, must be owned by root, and shared by all hosts of the same type
`$XLSF_APPDIR`	755	Window application resource files, shared by all hosts
`$XLSF_UIDDIR`	755	GUI UID files, shared by all hosts of the same type

C. LSF on Windows NT

This appendix describes how to run LSF on Windows NT. It is assumed that you are already familiar with LSF concepts, and have installed LSF on Windows NT following the instructions in the *LSF Installation Guide*.

Requirements

- You must use a domain account (as opposed to a local account) when interacting with .
- Users must enter their passwords into an encrypted database maintained by LSF and any changes to Windows NT passwords must be reflected in the password database used by LSF.

Recommended

- The Windows NT Resource Kit contains many useful utilities (for example, `pview`) for monitoring processes.
- A `telnet` daemon to enable remote login sessions or some other form of remote access software to allow for easier management.

Differences Between LSF for UNIX and NT

- The shell used to invoke commands is `cmd.exe` instead of `/bin/sh` as on UNIX. For example, the queue-level pre and post-exec commands are invoked as:

```
cmd.exe /C pre-exec command
```

- The NULL device on Windows NT is `NUL` rather than `/dev/null` as on UNIX. LSF translates `/dev/null` to `NUL` for Windows NT.

- The `/etc` directory on UNIX corresponds to the `%SYSTEMROOT%` directory on NT.

- On UNIX, LSF always uses `/tmp` as the temporary directory. On Windows NT, the temporary directory used by LSF can be configured by setting `LSF_TMPDIR` as a system environment variable. If that variable is not found, LSF goes to the next item in the following list, until a directory is defined:

  `LSF_TMPDIR` environment variable
  `LSF_TMPDIR` variable in the `lsf.conf` file
  `TMP` environment variable (`C:\temp` by default)
  `TEMP` environment variable (`C:\temp` by default)
  `%SYSTEMROOT%`

- There is no native support in Windows NT for UNIX-style signals. Therefore sending an arbitrary signal to a job via the `-s` option of `bkill` has no meaning on Windows NT. LSF, however, supports the job control functionality by providing the equivalent of `SIGSTOP`, `SIGCONT`, and `SIGTERM` to suspend, resume, and terminate a job. These can be accessed through the commands `bstop`, `bresume`, and `bkill`.

- The UNIX `umask` parameter is ignored on Windows NT.

- When inputting commands to `bsub`, remember that the syntax of the commands must be specified in the form understood by Windows NT batch files. For example to specify multiple commands in a single line, use '&&' as the command separator instead of '`;`' as in UNIX. For example, use:

  ```
  bsub ‘cmd1 && cmd2’
  ```

  instead of:

  ```
  bsub ‘cmd1; cmd2’
  ```

  Also when specifying commands from standard input, use CTRL-Z to indicate EOF. On UNIX, CTRL-D is used. For example:

```
c:\temp> bsub -q simulation
bsub> myjob arg1 arg2
bsub> ^Z
```

- The `tmp` index returned by `lim` measures the space on the drive specified by the `TEMP` system environment variable.

File Permissions

The directories `work`, `logs`, `bin`, `lib`, `etc`, and `conf`, are all subdirectories of your LSF directory.

For all LSF files, Platform Computing recommends that you give full control permission to the Domain Admins user group. Other permissions should be set as shown:

work, logs

LSF primary administrator:	full control (All) (All)
domain administrator:	full control (All) (All)
everyone:	special access (R) (R)

bin, lib, etc

LSF primary administrator:	full control (All) (All)
domain administrator:	full control (All) (All)
everyone:	special access (RX) (RX)

conf

LSF primary administrator:	full control (All) (All)
domain administrator:	full control (All) (All)

Mail

When LSF needs to send email to users, it invokes the program defined by LSB_MAILPROG in the `lsf.conf` file (in the `etc` subdirectory). If LSB_MAILPROG is not defined, no email is sent.

To use email, you need to use LSF's `lsmail.exe` program, which can send email to a UNIX host by using the Windows NT `rsh` utility (%WINDIR%\system32\rsh) to invoke `sendmail(1)` on the UNIX host. In order for this to work, the UNIX machine must be set up to allow the NT `rsh` client to run on it.

To support this method of sending email, `lsmail.exe` should be copied to a file corresponding to the name of the UNIX host. For example,

```
copy lsmail.exe unixhost.exe
```

Here `unixhost` is a UNIX machine which supports `sendmail(1)`. The LSB_MAILPROG should correspond to the `unixhost.exe` file. For example:

```
LSB_MAILPROG=//serverA/tools/lsf/bin/unixhost.exe
```

See *'LSB_MAILPROG'* on page 162 for details on how `LSB_MAILPROG` is invoked.

The `cmd.exe` Program

The command shell (`cmd.exe`) under Windows NT 4.0 cannot be started from a directory which is specified as a UNC name. For example, if you type the following on the command line, `cmd.exe` will end up starting in the directory specified by %WINDIR%, the system root directory of the current machine.

```
start /d\\serverA\share\username cmd.exe
```

As a result, jobs submitted from a shared directory will not start in the correct directory on the execution host.

The command shell from Windows NT 3.51, however, does support this feature. Microsoft has confirmed that this is a bug in NT 4.0, and included a fix in service pack 3 (refer to the article Q156276 in the Microsoft Knowledge Database for information).

In order for LSF to work correctly on Windows NT 4.0 machines, you can use one of three methods.

- Update your Windows NT 4.0 installation with service pack 3. LSF modifies the appropriate registry keys mentioned in article Q156276 to allow the UNC path to work.
- Replace your existing Windows NT 4.0 `cmd.exe` with the `cmd.exe` from service pack 3. The `cmd.exe` file typically resides in the `%WINDIR%\system32` directory. LSF modifies the appropriate registry keys mentioned in article Q156276 to allow the UNC path to work.
- Copy the Windows NT 3.51 `cmd.exe` into the `%WINDIR%\system32` directory under another name, e.g. `cmd351.exe`, and set the LSF_CMD_SHELL variable in the `lsf.conf` file to tell LSF to use this shell instead of `cmd.exe`.

  For example, put the following line into the `lsf.conf` file:

  ```
  LSF_CMD_SHELL=cmd351.exe
  ```

Heterogeneous NT/UNIX Environments

User Accounts

To run jobs in a mixed cluster, LSF users should have a user account with the same user name on UNIX and Windows NT. It is particularly important that the LSF primary administrator user account always have the same user name on both platforms.

Configuration Files

Note

If you used the Windows NT version of LSF Setup to create the UNIX/NT mixed cluster, as described in the LSF Installation Guide, the following settings have already been configured.

The LSF configuration files must be accessible from both NT and UNIX hosts. You can set up a shared file system between the UNIX and NT machines via NFS client on NT or an SMB server on UNIX, or, alternatively, you can replicate the configuration files. No matter how you arrange your configuration files, you must make sure that the port numbers (LSF_LIM_PORT, LSF_RES_PORT, LSF_SBD_PORT and LSF_MBD_PORT) defined in the `lsf.conf` file are the same on both UNIX and NT.

For example, if you use an SMB server on the UNIX side, you would simply set the three variables—LSF_CONFDIR, LSB_CONFDIR, and LSF_SHAREDIR—in the `lsf.conf` file to point to the corresponding directories used by the UNIX hosts. The LSF_CONFDIR and LSB_CONFDIR directories must be accessible to all users (read permission). However, only the LSF primary administrator should have full control of these directories (read and write permissions).

Environment Variables

By default, LSF transfers environment variables from the submission host to the execution host. However, some environment variables are not applicable to another operating system.

When submitting a job from a Windows NT machine to a UNIX machine, the `-L` option of the `bsub` command can be used to reinitialize the environment variables. If submitting a job from a UNIX machine to a NT machine, you can set the environment variables explicitly in your job script. Alternatively, the Job Starter feature can be used to reset the environment variables before starting the job. LSF automatically resets the PATH on the execution host if the submission host is of a different type.

If the submission host is Windows NT and the execution host is UNIX, then the PATH variable is set to `/bin:/usr/bin:/sbin:/usr/sbin` and LSF_BINDIR (if defined in the `lsf.conf` file) is appended to it. If the submission host is UNIX and the execution host is Windows NT, the PATH variable is set to the system PATH variable with LSF_BINDIR appended to it.

Cross-Platform Daemon Startup

The `lssrvcntrl.exe` binary only works when invoked from a Windows NT host. You will not be able to start LSF daemons on a Windows NT machine from a UNIX host.

Signal Conversion

LSF supports signal conversion between UNIX and Windows NT for remote interactive execution through RES (when you are using `lsrun` and `bsub -I`).

On Windows NT, the CTRL+C and CTRL+BREAK key combinations are treated as signals for console applications (these signals are also called console control events). LSF supports these two NT console signals for remote interactive execution, i.e. on the execution host LSF regenerates these signals for users' tasks. In a mixed NT/UNIX environment, LSF has the following default conversion between the NT console signals and the UNIX signals:

Table 2. UNIX/NT Signal Conversion

Windows NT	**UNIX**
CTRL+C	SIGINT
CTRL+BREAK	SIGQUIT

For example, if you issue `lsrun` or `bsub -I` commands from an NT console, but the task is running on a UNIX host, pressing the CTRL+C keys will generate a UNIX SIGINT signal to your task on the UNIX host. The reverse is also true.

Custom Signal Conversion

For `lsrun` (but not `bsub -I`), LSF allows users to define their own signal conversion using the following two environment variables.

- LSF_NT2UNIX_CLTRC
- LSF_NT2UNIX_CLTRB

For example, suppose a user sets the following:

```
LSF_NT2UNIX_CLTRC=SIGXXXX

LSF_NT2UNIX_CLTRB=SIGYYYY
```

Here, SIGXXXX/SIGYYYY are UNIX signal names such as SIGQUIT, SIGTTIN, etc. The conversions will then be: CTRL+C = SIGXXXX and CTRL+BREAK = SIGYYYY.

If both LSF_NT2UNIX_CLTRC and LSF_NT2UNIX_CLTRB are set to the same value, (LSF_NT2UNIX_CLTRC=SIGXXXX and LSF_NT2UNIX_CLTRB=SIGXXXX), then on the Windows NT execution host, CTRL+C will be generated.

For `bsub -I`, there is no conversion other than the default conversion.

Starting Services and Daemons

The LSF service and daemons on each LSF server host will start automatically when the machine is restarted.

If you cannot restart each host at this time, log on as an LSF cluster administrator (a member of the LSF Global Administrators group) and start the LSF service and daemons manually.

Note

You should not use the primary LSF administrator's account (normally `lsfadmin`*) to start or stop LSF service and daemons.*

To start the LSF service and daemons, use any one of the following methods:

- Use the Windows NT Server Manager to start “LSF Service” on all LSF server hosts.
- Click “Services” on the Windows NT Control Panel and start “LSF Service”. You will have to repeat this step on each LSF server host.
- Where LSF Batch has been installed, go to the “LSF Suite for Workload Management/LSF Batch” program folder, and use the LSF administrative tool

"LSF Batch Administration". (You can use this tool to perform all your administrative tasks for LSF Base and LSF Batch products.)

- Start a new command console, and type:

  ```
  lssrvcntrl start -m all lssrvman
  ```

  Usage information for `lssrvcntrl` is available by typing `lssrvcntrl` with no options.

Using LSF

Each user who wants to use LSF needs to supply the password of his / her domain user account. Use the `lspasswd.exe` command, and follow the instructions. For example:

```
lspasswd [-u user_name]
```

If you do not specify the `-u` option as above, the user is assumed to be the current user.

In addition, all users need to have the "Logon as a batch job" privilege on every LSF server host. For this purpose, you can simply put all LSF users into the 'LSF user group' created for or assigned by you during the installation. The LSF user group has the "Logon as a batch job" privilege on all LSF server hosts.

Miscellaneous

- The machines running LSF are expected to have fixed IP addresses. If you use DHCP to assign IP addresses dynamically, LSF can still work, provided the reassigned IP address of an LSF host does not change.

- When using LSF's remote execution functions through the Remote Execution Server, there is no support for `pty`-type options for `lsrun` and `bsub -I`, i.e. the `-P` and `-S` options for `lsrun` and `-Ip` and `-Is` options for `bsub` are not supported.

- If you log on as the LSF primary administrator from the console while the LSF service is running from a file server over the network, and then log off again, the LSF service and daemons may die on that host. Logging off appears to cause Windows NT to close all network connections for the LSF primary administrator user account, including those used by an LSF service or LSF daemons.

- When writing an external command that is invoked by LSF (for example, `elim`, `esub`, or `eexec`), the command must be a binary executable, that is, `elim.exe` or `esub.exe`. It cannot be a batch file such as `elim.bat`.

- `LSF_USE_HOSTEQUIV` parameter in `lsf.conf` is ignored on Windows NT.

- Nice values specified at the queue-level through the NICE parameter are mapped to NT process priority classes as follows:

  `nice>=0` corresponds to an NT priority class of `IDLE`
  `nice<0` corresponds to an NT priority class of `NORMAL`

  LSF does not support `HIGH` or `REAL-TIME` priority classes.

- The `io` index shows 0, unless the disk performance counters are turned on. To turn on disk performance counters, use the `DISKPERF` command.

  Note

  Turning on the performance counters incurs extra overhead in disk I/O.

- A job which runs under a CPU time limit may exceed that limit by up to `SBD_SLEEP_TIME`. This is because `sbatchd` periodically checks if the limit has been exceeded. On UNIX systems, the CPU limit can be enforced by the OS at the process level.

- The UNIX man pages, converted to HTML format, are stored in the `html` subdirectory of your `LSF` directory.

Index